

<u>DB Name</u>	<u>Query</u>	<u>Hit Count</u>	<u>Set Name</u>
USPT,PGPB	script and browser and (plug-in or (plug adj2 in)) and applet and (program near4 object)	29	<u>L21</u>
USPT	script and browser and (plug-in or (plug adj2 in)) and applet and (program near4 object)	24	<u>L20</u>
USPT	script and browser and (plug-in or (plug adj2 in)) and applet	84	<u>L19</u>
USPT	script and browser and (plug-in or (plug adj2 in))	181	<u>L18</u>
USPT	6266056.pn.	1	<u>L17</u>
EPAB,DWPI,TDBD	embedded near3 widget	2	<u>L16</u>
USPT,PGPB	embedded near3 widget	6	<u>L15</u>
USPT,PGPB	embedded near3 widget same script	0	<u>L14</u>
USPT	embedded near3 widget same script	0	<u>L13</u>
USPT	embedded near3 widget	6	<u>L12</u>
USPT,PGPB	embed\$ near6 object same script\$ same browser	12	<u>L11</u>
PGPB	embed\$ same script\$ same browser same widget	0	<u>L10</u>
PGPB	embed\$ same script\$ same browser	19	<u>L9</u>
USPT	embed\$ same script\$ same browser	124	<u>L8</u>
PGPB,JPAB,EPAB,DWPI,TDBD	embed\$ same script\$ same browser same widget	0	<u>L7</u>
USPT	embed\$ same script\$ same browser same widget	0	<u>L6</u>
USPT	(WAPI or (Web adj3 API)) same script\$	3	<u>L5</u>
USPT	WAPI or (Web adj3 API)	26	<u>L4</u>
USPT	(script near4 interpreter near4 extension)	2	<u>L3</u>
USPT	(script near4 interpreter) same (browser near5 (plug-in or (plug adj2 in))) same (object or applet or scriptlet)	0	<u>L2</u>
USPT	script same browser same pars\$ same (plug-in or (plug adj2 in))	4	<u>L1</u>

WEST Generate Collection

L11: Entry 7 of 12

File: USPT

Dec 12, 2000

DOCUMENT-IDENTIFIER: US 6161126 A

TITLE: Implementing force feedback over the World Wide Web and other computer networks

BSPR:

Furthermore, additional functionality may be provided in web pages with information downloaded over the WWW in the form of scripts or programs. Scripting, typically in the form of VBScript or JavaScript, allows a series of instructions to be performed on the client computer once the instructions have been downloaded in a web page. Programs can be provided in such standard languages as Visual Basic, C++, or currently in the form of Java "applets" or ActiveX.RTM. controls. Java includes a platform-independent interpreter running on the client machine that executes downloaded applets within web pages or other programs, e.g., to display animated images, retrieve data over the WWW, output feedback to the user, or perform operating system tasks. ActiveX controls similarly execute on the client computer once the program instructions are resident on the client. ActiveX controls are programmable objects that can be embedded into Web pages and may be written in any (platform-specific) language. Java and ActiveX controls can add functionality to a Web page that would normally be difficult, or, even impossible, using HTML or scripting languages. ActiveX controls can also be controlled with a scripting language. Alternatively, functionality can be added to a web page through the use of "plug-ins", which are application programs running in conjunction with certain web browsers to parse plug-in-specific code in the web page which the browser cannot understand.

WEST

Generate Collection

L11: Entry 5 of 12

File: USPT

Feb 13, 2001

DOCUMENT-IDENTIFIER: US 6188401 B1

TITLE: Script-based user interface implementation defining components using a text markup language

BSPR:

The invention utilizes a supervisory application program that runs under an operating system such as Windows CE. The application program implements an extended document object model for use by the global script and by scripts embedded in HTML control elements. The global script runs perpetually, under the supervision of the supervisory application program, as does one or more instances of a conventional Web browser.

WEST

Generate Collection

L11: Entry 4 of 12

File: USPT

Feb 13, 2001

DOCUMENT-IDENTIFIER: US 6189000 B1

TITLE: System and method for accessing user properties from multiple storage mechanisms

BSPR:

Software object components also may be used with the HTML documents for displaying executable content, such as for animations or information processing. Currently, most Internet browsers support embedded software object components in the form of ActiveX controls, Java applets, and scripts (e.g., VB scripts and Java scripts).

WEST **Generate Collection**

LS5: Entry 3 of 3

File: USPT

Mar 16, 1999

DOCUMENT-IDENTIFIER: US 5884312 A

TITLE: System and method for securely accessing information from disparate data sources through a network

DEPR:

The web server program allows web browser-enabled clients of the Internet or web browser-enabled clients of a corporate intranet to receive graphical documents that are either stored at web browser 28 and identified by a specific URL or generated using a particular script at web browser 28 also identified by a specific URL. The scripts are computer programs stored in a scripting language or programming language that may be implemented according to the Common Gateway Interface (CGI) standard or the NETSCAPE SERVER APPLICATION PROGRAM INTERFACE (NSAPI) or similar web server API that describes how web servers of the WWW should access external programs so that data is returned to the client in the form of an automatically generated web page. Scripts are normally needed when the user fills out onscreen forms which the script uses as an input to bring about the execution of other programs as needed. Ultimately, the scripts generate a web page to provide as an output to the web browser-enabled client. The web pages will generally be provided as a text file encoded with a declarative markup language (DML) such as the Standard Generalized Markup Language (SGML) or, preferably, with HyperText Markup Language (HTML).

WEST

Generate Collection

L11: Entry 3 of 12

File: USPT

Apr 10, 2001

DOCUMENT-IDENTIFIER: US 6216141 B1

TITLE: System and method for integrating a document into a desktop window on a client computer

BSPR:

Software object components also may be used with the HTML document for displaying executable content, such as for animations or information processing. Currently, most Internet browsers support embedded software object components in the form of ActiveX controls, Java applets, and Visual Basic Scripts. These software object components are inserted into HTML documents using the or HTML tags.

WEST **Generate Collection**

L21: Entry 12 of 29

File: USPT

May 22, 2001

DOCUMENT-IDENTIFIER: US 6237006 B1

TITLE: Methods for graphically representing web sites and hierarchical node structures

ABPL:

A visual Web site analysis program, implemented as a collection of software components, provides a variety of features for facilitating the analysis and management of web sites and Web site content. A mapping component scans a Web site over a network connection and builds a site map which graphically depicts the URLs and links of the site. Site maps are generated using a unique layout and display methodology which allows the user to visualize the overall architecture of the Web site. Various map navigation and URL filtering features are provided to facilitate the task of identifying and repairing common Web site problems, such as links to missing URLs. A dynamic page scan feature enables the user to include dynamically-generated Web pages within the site map by capturing the output of a standard Web browser when a form is submitted by the user, and then automatically resubmitting this output during subsequent mappings of the site. The Web site analysis program is implemented using an extensible architecture which includes an API that allows plug-in applications to manipulate the display of the site map. Various plug-ins are provided which utilize the API to extend the functionality of the analysis program, including an action tracking plug-in which detects user activity and behavioral data (link activity levels, common site entry and exit points, etc.) from server log files and then superimposes such data onto the site map.

BSPR:

In accordance with another aspect of the invention, the Web site analysis program is based on an extensible architecture that allows software components to be added that make extensive use of the program's mapping functionality. Specifically, the architecture includes an API (application program interface) which includes API procedures ("methods") that allow other applications ("plug-ins") to, among other things, manipulate the display attributes of the nodes and links within a site map. Using these methods, a plug-in application can be added which dynamically superimposes data onto the site map by, for example, selectively modifying display colors of nodes and links, selectively hiding nodes and links, and/or attaching alphanumeric annotations to the nodes and links. The API also includes methods for allowing plug-in components to access Web site data (both during and following the Web site scanning process) retrieved by the scanning routines, and for adding menu commands to the user interface of the main program.

BSPR:

In accordance with another aspect of the invention, software routines (preferably implemented within a plug-in application) are provided for processing a Web site's server access log file to generate Web site usage data, and for displaying the usage data on a site map. This usage data may, for example, be in the form of the number of "hits" per link, the number of Web site exit events per node, or the navigation paths taken by specific users ("visitors"). This usage data is preferably generated by processing the entries within the log file on a per-visitor basis to determine the probable navigation path taken by each respective visitor to the Web site. (Standard-format access log files which record each access to any page of the Web site are typically maintained by conventional Web servers.) In a preferred implementation, the usage data is then superimposed onto the site map (using the API methods) using different node and link display colors to represent different respective levels of user activity. Using this feature, Webmasters can readily detect common "problem areas" such as congested links and popular Web site exit points. In addition, by looking at individual navigation paths on a per-visitor basis, Webmasters can identify popular navigation paths taken by visitors to the site.

BSPR:

To effectuate the capture of one or more datasets in the preferred implementation, the user initiates a capture session from the user interface; this causes a standard Web browser to be launched and temporarily configured to use the Web site analysis program as an HTTP-level proxy to communicate with Web sites. Thereafter, until the capture session is terminated by the user, any pages retrieved with the browser, and any forms (datasets) submitted from the browser, are automatically recorded by the Web site analysis program into the site map. When the site map is subsequently updated (using an "automatic update" option of the user interface), the scanning routines automatically re-enter the captured datasets into the corresponding forms and recreate the form submissions. The dynamically-generated Web pages returned in response to these automatic form submissions are then added to the updated site map as respective nodes. A related aspect of the invention involves the associated method of locally capturing the output of the Web browser to generate a sequence that can subsequently be used to automatically evaluate a Web site.

DEPR:

Document. Generally, a collection of data that can be viewed using an application program, and that appears or is treated as a self-contained entity. Documents typically include control codes that specify how the document content is displayed by the application program. An "HTML document" is a special type of document which includes HTML (HyperText Markup Language) codes to permit the document to be viewed using a Web browser program. An HTML document that is accessible on a World Wide Web site is commonly referred to as a "Web document" or "Web page." Web documents commonly include embedded components, such as GIF (Graphics Interchange Format) files, which are represented within the HTML coding as links to other URLs. (See "HTML" and "URL" below.)

DEPR:

World Wide Web. A distributed, global hypertext system, based on an set of standard protocols and conventions (such as HTTP and HTML, discussed below), which uses the Internet as a transport mechanism. A software program which allows users to request and view World Wide Web ("Web") documents is commonly referred to as a "Web browser," and a program which responds to such requests by returning ("serving") Web documents is commonly referred to as a "Web server."

DEPR:

Content Object. As used herein, a data entity (document, document component, etc.) that can be selectively retrieved from a web site. In the context of the World Wide Web, common types of content objects include HTML documents, GIF files, sound files, video files, Java applets and aglets, and downloadable applications, and each object has a unique identifier (referred to as the "URL") which specifies the location of the object. (See "URL" below.)

DEPR:

HTML (HyperText Markup Language). A standard coding convention and set of codes for attaching presentation and linking attributes to informational content within documents. During a document authoring stage, the HTML codes (referred to as "tags") are embedded within the informational content of the document. When the Web document (or "HTML document") is subsequently transmitted by a Web server to a Web browser, the codes are interpreted by the browser and used to parse and display the document. In addition to specifying how the Web browser is to display the document, HTML tags can be used create hyperlinks to other Web documents. For more information on HTML, see Ian S. Graham, The HTML Source Book, John Wiley and Sons, Inc., 1995 (ISBN 0471-11894-4).

DEPR:

HTTP (Hypertext Transfer Protocol). The standard World Wide Web client-server protocol used for the exchange of information (such as HTML documents, and client requests for such documents) between a Web browser and a Web server. HTTP includes several different types of messages which can be sent from the client to the server to request different types of server actions. For example, a "GET" message, which has the format GET , causes the server to return the content object located at the specified URL.

DEPR:

CGI (Common Gateway Interface). A standard interface which specifies how a Web server (or possibly another information server) launches and interacts with

external programs (such as a database search engine) in response to requests from clients. With CGI, the Web server can serve information which is stored in a format that is not readable by the client, and present such information in the form of a client-readable Web page. A CGI program (called a "CGI script") may be invoked, for example, when a Web user fills out an on-screen form which specifies a database query. For more information on CGI, see Ian S. Graham, *The HTML Source Book*, John Wiley and Sons, Inc., 1995 (ISBN 0471-11894-4), pp. 231-278.

DEPR:

OLE (Object Linking and Embedding). An object technology, implemented by Windows-based applications, which allows objects to be linked to one another and embedded within one another. OLE Automation, which is a feature of OLE 2, enables a program's functionality to be exposed as OLE objects that can be used to build other applications. For additional information on OLE and OLE Automation, see OLE 2 Programmer's Reference Manual, Volume One, Microsoft Corporation, 1996 (ISBN 1-55615-628-6).

DEPR:

Given the address of a Web site's home page, Astra automatically scans the Web site and creates a graphical site map showing all of the URLs of the site and the links between these URLs. In accordance with one aspect of the invention, the layout and display method used by Astra for generating the site map provides a highly intuitive, graphical representation which allows the user to visualize the layout of the site. Using this mapping feature, in combination with Astra's powerful set of integrated tools for navigating, filtering and manipulating the Web site map, users can intuitively perform such actions as isolate and repair broken links, focus in on Web pages (and other content objects) of a particular content type and/or status, and highlight modifications made to a Web site since a prior mapping. In addition, users can utilize a Dynamic Scan.TM. feature of Astra to automatically append dynamically-generated Web pages (such as pages generated using CGI scripts) to their maps. Further, using Astra's activity monitoring features, users can monitor visitor activity levels on individual links and URLs, and study visitor behavior patterns during Web site visits.

DEPR:

In accordance with another aspect of the invention, Astra has a highly extensible architecture which facilitates the addition of new tools to the Astra framework. As part of this architecture, a "core" Astra component (which includes the basic Web site scanning and mapping functionality) has an API for supporting the addition of plug-in components. This API includes functions for allowing the plug-in components to manipulate the display of the site map, and to display their own respective data in conjunction with the Astra site map. Through this API, new applications can be added which extend the functionality of the package while taking advantage of the Astra mapping scheme.

DEPR:

FIG. 1 illustrates a site map 30 of a demonstration Web site which was derived from the actual Web site of Mercury Interactive, Inc. (i.e., the URLs accessible under the "merc-int.com" Internet domain name). (For purposes of this detailed description, it may be assumed that "Web site" refers to the content associated with a particular Internet domain name.) The Web site is depicted by Astra as a collection of nodes, with pairs of nodes interconnected by lines. Each node of the map represents a respective content object of the Web site and corresponds to a respective URL. (The term "URL" is used herein to refer interchangeably to both the address of the content object and to the object itself; where a distinction between the two is helpful to an understanding of the invention, the term "URL" is followed by an explanatory parenthetical.) Examples of URLs (content objects) which may exist within a typical Web site include HTML documents (also referred to herein as "Web pages"), image files (e.g., GIF and PCX files), mail messages, Java applets and aglets, audio files, video files, and applications.

DEPR:

The lines which interconnect the nodes (URL icons) in FIGS. 1-3 (and the subsequent figures with screen displays) represent links between URLs. As is well understood in the art, the functions performed by these links vary according to URL type. For example, a link from one HTML document to another HTML document normally represents a hyperlink which allows the user to jump from one document to the other while navigating the Web site with a browser. In FIG. 1, an example of a hyperlink which links the home page URL (shown at the center of the map) to

another HTML page (displayed to the right of the home page) is denoted by reference number 32. (As generally illustrated in FIG. 1 and the other figures which illustrate screen displays, regular HTML documents are displayed by Astra as a shaded document having text thereon.) A link between an HTML document and a GIF file, such as link 36 in FIG. 3, normally represents a graphic which is embedded within the Web page.

DEPR:

Maps of the type illustrated in FIG. 1 are generated by Astra using an HTTP-level scanning process (described below) which involves the reading and parsing the Web site's HTML pages to identify the architecture (i.e., the arrangement of URLs and links) of the Web site, and to obtain various status information (described below) about the Web site's URLs. The basic scanning process used for this purpose is generally similar to the scanning process used by conventional Webcrawlers. As part of Astra's Dynamic Scan feature, Astra additionally implements a special dynamic page scanning process which permits dynamically-generated Web pages to be scanned and included in the Web site map. As described below, this process involves capturing the output of a Web browser when the user submits an HTML-embedded form (such as when the user submits a database query), and then reusing the captured dataset during the scanning process to recreate the form submission and append the results to the map.

DEPR:

While navigating the map, the user can retrieve a URL (content object) from the server by double-clicking on the corresponding URL icon; this causes Astra to launch the client computer's default Web browser (if not already running), which in-turn retrieves the URL from the Web server. For example, the user can double-click on the URL icon for an HTML document (using the left mouse button) to retrieve and view the corresponding Web page. When the user clicks on a URL icon using the right mouse button, a menu appears which allows the user to perform a variety of actions with respect to the URL, including viewing the URL's properties, and launching an HTML editor to retrieve and edit the URL. With reference to FIG. 3, for example, the user can click on node 44 (using the right mouse button), and can then launch an HTML editor to edit the HTML document and delete the reference to missing URL 45. (As illustrated by FIG. 3, missing URLs are represented within Astra maps by a question mark icon.)

DEPR:

Another benefit of this site map layout and display methodology is that the resulting display structure is well suited for the overlaying of information on the map. Astra takes full advantage of this benefit by providing a set of API functions which allow other applications (Astra plug-ins) to manipulate and add their respective display data to the site map. An example of an Astra plug-in which utilizes this feature is the Action Tracker.TM. tool, which superimposes user activity data onto the site map based on analyses of server access log files. The Astra plug-in API and the Action Tracker plug-in are described in detail below.

DEPR:

As illustrated in FIG. 1, the Astra menu bar includes seven menu headings: FILE, VIEW, SCAN, MAP, URL, TOOLS and HELP. From the FILE menu the user can perform various file-related operations, such as save a map file to disk or open a previously generated map file. From the VIEW menu the user can select various display options of the Astra GUI. From the SCAN menu the user can control various scanning-related activities, such as initiate or pause the automatic updating of a map, or initiate a dynamic page scan session. From the MAP menu, the user can manipulate the display of the map, by, for example, collapsing (hiding) all leaf nodes, or selecting the Visual Web Display mode. From the URL menu, the user can perform operations with respect to user-selected URLs, such as display the URL's content with a browser, invoke an editor to modify the URL's content, and display the incoming or outgoing links to/from the URL.

DEPR:

FIG. 7 pictorially illustrates the general architecture of Astra, as installed on a client computer 92. As illustrated, the architecture generally consists of a core Astra component 94 which communicates with a variety of different Astra plug-in applications 96 via a plug-in API 98. The Astra core 94 includes the basic functionality for the scanning and mapping of Web sites, and includes the above-described GUI features for facilitating navigation of Web site maps. Through

the plug-in API 98, the Astra core 94 provides an extensible framework for allowing new applications to be written which extend the basic functionality of the Astra core. As described below, the architecture is structured such that the plug-in applications can make extensive use of Astra site maps to display plug-in specific information.

DEPR:

The Astra plug-ins 96 and API 98 are based on OLE Automation technology, which provides facilities for allowing the plug-in components to publish information to other objects via the operating system registry (not shown). (The "registry" is a database used under the Windows.RTM. 95 and Windows.RTM. NT operating systems to store configuration information about a computer, including information about Windows-based applications installed on the computer.) At start-up, the Astra core 94 reads the registry to identify the Astra plug-ins that are currently installed on the client computer 92, and then uses this information to launch the installed plug-ins.

DEPR:

In a preferred implementation, the architecture includes five Astra plug-ins: Link Doctor, Action Tracker, Test World, Load Wizard and Search Meter. The functions performed by these plug-ins are summarized by Table 2. Other applications which will normally be installed on the client computer in conjunction with Astra include a standard Web browser (FIGS. 11 and 12), and one or more editors (not shown) for editing URL content.

DEPR:

The Astra API allows external client applications, such as the plug-in applications 96 shown in FIG. 7, to communicate with the Astra core 94 in order to form a variety of tasks. Via this API, client applications can perform the following types of operations:

DEPR:

As is conventional with Internet applications, the Astra core 94 uses the TCP/IP layer 108 of the computer's operating system to communicate with the Web site 113. Any one or more of the Astra plug-ins 96 may also use the TCP/IP layer 108 to communicate with the Web site 113. In the preferred embodiment, for example, the Action Tracker plug-in communicates with the Web sites (via the Astra plug-in API) to retrieve server access log files for performing Web site activity analyses.

DEPR:

Each Node object 115 represents a respective node (URL) of the site map, and each Edge object 116 represents a respective link between two URLs (nodes) of the map. Associated with each Node object and each Edge object is a set of attributes (not shown), including display attributes which specify how the respective object is to be represented graphically within the site map. For example, each Node object and each Edge object include respective attributes for specifying the color, visibility, size, screen position, and an annotation for the display of the object. These attributes can be manipulated via API calls to the methods supported by these objects 115, 116. For example, the Astra plug-ins (FIG. 7) can manipulate the visibility attributes of the Edge objects to selectively hide the corresponding links on the screen. (This feature is illustrated below in the description of the Action Tracker plug-in.) In addition, the Astra API includes methods for allowing the plug-ins to define and attach custom attributes to the Edge and Node objects.

DEPR:

The methods of the Astra plug-in API generally fall into five functional categories. These categories, and the objects to which the associated methods apply, are listed below. Additional information on these methods is provided in the API listing in Appendix B.

DEPR:

As will be appreciated from the foregoing, the Astra architecture provides a highly extensible mapping framework which can be extended in functionality by the addition of new plug-ins applications. Additional aspects of the architecture are specified in the API description of Appendix B.

DEPR:

A feature of the invention which permits the scanning and mapping of

dynamically-generated Web pages will now be described. By way of background, a dynamically-generated Web page ("dynamic page") is a page that is generated "on-the-fly" by a Web site in response to some user input, such as a database query. Under existing Web technology, the user manually types-in the information (referred to herein as the "dataset") into an embedded form of an HTML document while viewing the document with a Web browser, and then selects a "submit" type button to submit the dataset to a Web site that has back-end database access or real-time data generation capabilities. (Technologies which provide such Web server extension capabilities include CGI, Microsoft's ISAPI, and Netscape's NSAPI.) A Web server extension module (such as a CGI script) then processes the dataset (by, for example, performing a database search, or generating real-time data) to generate the data to be returned to the user, and the data is returned to the browser in the form of a standard Web page.

DEPR:

One deficiency in existing Web site mapping programs is that they do not support the automatic retrieval of dynamic pages. As a result, these mapping programs are not well suited for tracking changes to back-end databases, and do not provide an efficient mechanism for testing the functionality of back-end database search components. The present invention overcomes these deficiencies by providing a mechanism for capturing datasets entered by the user into a standard Web browser, and for automatically re-submitting such datasets during the updating of site maps. The feature of Astra which provides these capabilities is referred to as Dynamic Scan.TM..

DEPR:

FIG. 11 illustrates the general flow of information between components during a Dynamic Scan capture session, which can be initiated by the user from the Astra tool bar. Depicted in the drawing is a client computer 92 communicating with a Web site 113 over the Internet 110 via respective TCP/IP layers 108, 178. The Web site 113 includes a Web server application 112 which interoperates with CGI scripts (shown as layer 180) to generate Web pages on-the-fly. Running on the client computer 92 in conjunction with the Astra application 94 is a standard Web browser 170 (such as Netscape Navigator or Microsoft's Internet Explorer), which is automatically launched by Astra when the user activates the capture session. As illustrated, the Web browser 170 is configured to use the Astra application 94 as an HTTP-level proxy. Thus, all HTTP-level messages (client requests) generated by the Web browser 170 are initially passed to Astra 94, which in-turn makes the client requests on behalf of the Web browser. Server responses (HTML pages, etc.) to such requests are returned to Astra by the client computer's TCP/IP layer 108, and are then forwarded to the browser to maintain the impression of normal browsing.

DEPR:

During the Dynamic Scan capture session, the user types-in data into one or more fields 174 of an HTML document 172 while viewing the document with the browser 170. The HTML document 172 may, for example, be an internal URL which is part of a Web site map, or may be an external URL which has been linked to the site map for mapping purposes. When the user submits the form, Astra extracts the manually-entered dataset, and stores this dataset (in association with the HTML document 172) for subsequent use. When Astra subsequently re-scans the HTML document 172 (during an Automatic Update of the associated site map), Astra automatically retrieves the dataset, and submits the dataset to the Web site 113 to recreate the form submission. Thus, for example, once the user has typed-in and submitted a database query in connection with a URL of a site map, Astra will automatically perform the database query (and map the results, as described below) the next time an Automatic Update of the map is performed.

DEPR:

With further reference to FIG. 11, when the Web site 113 returns the dynamic page during the capture session (or during a subsequent Automatic Update session), Astra automatically adds a corresponding node to the site map, with this node being displayed as being linked to the form page. (Screen displays taken during a sample capture session are shown in FIGS. 13-15 and are described below.) In addition, Astra parses the dynamic page, and adds respective nodes to the map for each outgoing link of the dynamic page. (In the default setting, these outgoing links are not scanned.) Astra also parses any static Web pages that are retrieved with the browser during the Dynamic Scan capture session, and updates the site map (by appending appropriate URL icons) to reflect the static pages.

DEPR:

Prior to initiating the Dynamic Scan session, the user specifies a page 172 which includes an embedded form. (This step is not shown in FIG. 12). This can be done by browsing the site map with the Astra GUI to locate the node of a form page 172 (depicted by Astra using a special icon), and then selecting the node with the mouse. The user then initiates a Dynamic Scan session, which causes the following dialog to appear on the screen: YOU ARE ABOUT TO ENTER DYNAMIC SCAN MODE. IN THIS MODE YOU WORK WITH A BROWSER AS USUAL, BUT ALL YOUR ACTIONS (INCLUDING FORM SUBMISSIONS) ARE RECORDED IN THE SITE MAP. TO EXIT FROM THIS MODE, PRESS THE "STOP DYNAMIC SCAN" BUTTON ON THE MAIN TOOLBAR OR CHOOSE THE "STOP DYNAMIC SCAN" OPTION IN THE SCAN MENU.

DEPR:

When the user clicks on the "OK" button, Astra modifies the configuration of the Web browser 170 within the registry 182 of the client computer to set Astra 94 as a proxy of the browser, as illustrated by arrow A of FIG. 12. (As will be recognized by those skilled in the art, the specific modification which needs to be made to the registry 182 depends upon the default browser installed on the client computer.) Astra then launches the browser 170, and passes the URL (address) of the selected form page to the browser for display. Once the browser has been launched, Astra modifies the registry 182 (arrow B) to restore the original browser configuration. This ensures that the browser will not attempt to use Astra as a proxy on subsequent browser launches, but does disable the browser's use of Astra as a proxy during the Dynamic Scan session.

DEPR:

As depicted in FIG. 12, the browser 170 retrieves and displays the form page 172, enabling the user to complete the form. In response to the submission by the user of the form, the browser 170 passes an HTTP-level (GET or POST) message to Astra 94, as indicated by arrow C. This message includes the dataset entered by the user, and specifies the URL (address) of the CGI script or other Web server extension component 180 to which the form is addressed. Upon receiving this HTTP message, Astra displays the dialog "YOU ARE ABOUT TO ADD A DATA SET TO THE CURRENT URL IN THE SITE MAP," and presents the user with an "OK" button and a "CANCEL" button.

DEPR:

Assuming the user selects the OK button, Astra extracts the dataset entered by the user and then forwards the HTTP-level message to its destination, as illustrated by arrow E. In addition, as depicted by arrow D, Astra stores this dataset in the Site Graph 114 in association with the form page 172. As described above, this dataset will automatically be retrieved and re-submitted each time the form page 172 is re-scanned as part of an Automatic Update operation. With reference to arrows F and G, when the Web server 112 returns the dynamic page 184, Astra 94 parses the page and updates the Site Graph 114 to reflect the page and any outgoing links of the dynamic page. (In this regard, Astra handles the dynamic page in the same manner as for other HTML documents retrieved during the normal scanning process.) In addition, as depicted by arrow H, Astra forwards the dynamic page 184 to the Web browser 170 (which in-turn displays the page) to maintain an impression of normal Web browsing.

DEPR:

Following the above sequence, the user can select the "stop dynamic scan" button or menu option to end the capture session and close the browser 170. Alternatively, the user can continue the browsing session and make additional updates to the site map. For example, the user can select the "back" button 186 (FIG. 14) of the browser to go back to the form page and submit a new dataset, in which case Astra will record the dataset and resulting page in the same manner as described above.

DEPR:

Although the system of the preferred embodiment utilizes conventional proxy technology to redirect and monitor the output of the Web browser 170, it will be recognized that other technologies and redirection methods can be used for this purpose. For example, the output of the Web browser could be monitored using conventional Internet firewall technologies.

DEPR:

FIG. 14 illustrates a subsequent screen display generated by starting a Dynamic Scan session with the Infoseek.TM. page selected, and then typing in the word "school" into the query field 194 of the page. (Intermediate displays generated by Astra during the Dynamic Scan session are omitted.) As illustrated in the figure, the Web browser comes up within a window 196, allowing the user to access the Astra controls and view the site map 190 during the Dynamic Scan session.

DEPR:

As generally illustrated by FIG. 15, in which the children 204 of the dynamic page 200 are represented with Astra's "not scanned," Astra does not automatically scan the children of the dynamically-generated Web page during the Dynamic Scan session. To effectively scan a child page 204, the user can retrieve the page with the browser during the Dynamic Scan session, which will cause Astra to parse the child page and update the map accordingly.

DEPR:

While the above-described Dynamic Scan feature is particularly useful in Web site mapping applications, it will be recognized that the feature can also be used to in other types of applications. For example, the feature can be used to permit the scanning of dynamically-generated pages by general purpose Webcrawlers. In addition, although the feature is implemented in the preferred embodiment such that the user can use a standard, stand-alone Web browser, it will be readily apparent that the feature can be implemented using a special "built-in" Web browser that is integrated with the scanning and mapping code.

DEPR:

An important feature of Astra is its the ability to track user (visitor) activity and behavior patterns with respect to a Web site and to graphically display this information (using color coding, annotations, etc.) on the site map. In the preferred embodiment, this feature is implemented in-part by the Action Tracker plug-in, which gathers user activity data by retrieving and analyzing server log files commonly maintained by Web servers. Using this feature, Webmasters can view site maps which graphically display such information as: the most frequently-accessed URLs, the most heavily traveled links and paths, and the most popular site entry and exit points. As will be appreciated by those skilled in the art, the ability to view such information in the context of a site map greatly simplifies the task of evaluating and maintaining Web site effectiveness.

DEPR:

FIG. 19 illustrates the general display format used by the Action Tracker plug-in to display activity levels on the links of a site. As illustrated by the screen display, the links between URLs are displayed using a color-coding scheme which allows the user to associate different link colors (and URL icon colors) with different relative levels of user activity. As generally illustrated by the color legend, three distinct colors are used to represent three (respective) adjacent ranges of user activity.

DEPR:

FIG. 20 illustrates the general process used by the Action Tracker plug-in to detect the link activity data (number of hits per link) from the log file. The displayed flow chart assumes that the log file has already been retrieved, and that the attribute "hits" has been defined for each link (Edge object) of the Site Graph and set to zero. As illustrated by the flow chart, the general decision process is applied line-by-line to the log file (each line representing an access to a URL) until all of the lines have been processed. With reference to blocks 250 and 252, each time a new line of the log file is ready, it is initially determined whether or not the log file reflects a previous access by the user to the Web site. This determination is made by searching for other entries within the log file which have the same user identifier (e.g., IP address) and an earlier date/time stamp.

DEPR:

Blocks 254 and 256 illustrate the steps that are performed if the user (visitor) previously visited the site. Initially, the Site Graph is accessed to determine whether a link exists from the most-recently accessed URL to the current URL, as indicated by decision block 254. If such a link exists, it is assumed that the visitor used this link to get to the current URL, and the usage level ("hits" attribute) of the identified link is incremented by one. If no such link is identified between the most-recently accessed URL and the current URL, an

assumption is made that the user back-tracked along the navigation path (by using the "BACK" button of the browser) before jumping to the current URL. Thus, decision step 254 is repeated for each prior access by the user to the site, in reverse chronological order, until either a link to the current URL is identified or all of the prior accesses are evaluated. If a link is detected during this process, the "hits" attribute of the link is incremented.

DEPR:

Usage Zones. When viewing a large site map in its entirety (as in FIG. 1), it tends to be difficult to identify individual URL icons within the map. This in-turn makes it difficult to view the color-coding scheme used by the Action Tracker plug-in to display URL usage levels. The Usage Zones.TM. feature alleviates this problem by enlarging the size of the colored URL icons (i.e., the icons of nodes which fall within the predetermined activity level thresholds) to a predetermined minimum size. (This is accomplished by increasing the "display size" attributes of these icons.) If these colored nodes are close together on the map, the enlarged icons merge to form a colored zone on the map. This facilitates the visual identification of high-activity zones of the site.

DEPR:

FIG. 22 illustrates the operation of Astra's Link Doctor plug-in. To access this feature, the user selects the "Link Doctor" option from the TOOLS menu while viewing a site map. The Link Doctor dialog box 284 then appears with a listing (in the "broken links" pane 286) of all of the broken links (i.e., URLs of missing content objects) detected within the site map. (Astra detects the missing links by searching the Site Graph for Node objects having a status of "not found.") When the user selects a URL from the broken links pane (as illustrated in the screen display), Astra automatically lists all of the URLs which reference the missing content object in the "appearing in" pane 288. This allows the user to rapidly identify all of the URLs (content objects) that are directly affected by the broken link.

DEPR:

While certain preferred embodiments of the invention have been described, these embodiments have been presented by way of example only, and are not intended to limit the scope of the present invention. For example, although the present invention has been described with reference to the standard protocols, services and components of the World Wide Web, it should be recognized that the invention is not so limited, and that the various aspects of the invention can be readily applied to other types of web sites, including intranet sites and network sites that use proprietary client-server protocols. In addition, while the Web site mapping and map navigation features of the invention have been described in the context of a preferred Web site management and analysis program, these features can also be included within browser programs to facilitate the visualization and navigation of Web sites by visitors. Accordingly, the breadth and scope of the present invention should be defined only in accordance with the following claims and their equivalents.

DEPV:

XI. Link Repair Plug-in

DETL:

TABLE 2 PLUG-IN FUNCTION PERFORMED Link Fixes broken links automatically Doctor Action Retrieves and evaluates server log files to generate Web site Tracker activity data (such as activity levels on individual links), and superimposes such data on site map in a user-adjustable manner. Test Generates and drives tests automatically World Load Utilizes site map to automatically generate test scripts for the Wizard load testing of Web sites with Mercury Interactive's LoadRunner .TM. and SiteTest .TM. software packages. Search Displays search engine results visually Meter

WEST **Generate Collection**

L21: Entry 13 of 29

File: USPT

Apr 10, 2001

DOCUMENT-IDENTIFIER: US 6216141 B1

TITLE: System and method for integrating a document into a desktop window on a client computer

BSPR:

Software object components also may be used with the HTML document for displaying executable content, such as for animations or information processing. Currently, most Internet browsers support embedded software object components in the form of ActiveX controls, Java applets, and Visual Basic Scripts. These software object components are inserted into HTML documents using the or HTML tags.

BSPR:

In still a further aspect of the invention, content providers not listed in the channel guide are able to provide their own channel guide and display documents on the desktop of the client computer. Special controls (ActiveX Controls, Java applets, VB Scripts, etc.) are used by content providers not listed in the channel guide to display the documents. The control allows for storage of a URL corresponding to the content providers in the user-preference storage.

BSPR:

Thus, using the present invention, rich multimedia documents are integrated directly into the desktop window on a client computer. The documents are HTML and may include VB scripts, java applets and hyperlinkable content allowing the user to browse a computer network, such as the Internet or Intranet. Further, the present invention allows content providers listed and content providers not listed in the channel guide to display documents in the desktop window. Finally, the content displayed is the most up-to-date content, since it is provided directly from the content providers.

DRPR:

FIG. 3 is a known browser environment on a client computer for connecting to and interacting with an Internet server computer.

DRPR:

FIG. 6 is an illustration of a browser for displaying additional information about the document displayed in the desktop viewer of FIG. 5.

DEPR:

An object is an instance of a programmer-defined type referred to as a class, which exhibits the characteristics of data encapsulation, polymorphism and inheritance. Data encapsulation refers to the combining of data (also referred to as properties of an object) with methods that operate on the data (also referred to as member functions of an object) into a unitary software component (i.e., the object), such that the object hides its internal composition, structure and operation and exposes its functionality to client programs that utilize the object only through one or more interfaces. An interface of the object is a group of semantically related member functions of the object. In other words, the client programs do not access the object's data directly, but must instead call functions on the object's interfaces to operate on the data.

DEPR:

The pointer 60, the virtual function table 54, and the member functions 56-58 implement an interface of the object 50. Client programs interact with the object 50 by obtaining a pointer (referred to as an interface pointer) to the pointer 60 of the virtual function table 54. OLE includes a type definition of an interface pointer which allows client programs to call member functions on the interface by name through the interface pointer and provides type checking on the function's arguments, as expressed in the following code (in the C++ programming language):

DEPR:

The object 50 conforming to the COM specification exhibits data encapsulation by exposing its interfaces (semantic groupings of its member functions) to client programs. The client programs interact with the object 50 by calling the member functions 56-58 on a particular interface of the object, but do not directly manipulate the object's data. The object 50 also exhibits polymorphism and inheritance in that the object 50 can provide interfaces in common with a base class and other similar objects, so that client programs can interact with each of the objects in the same manner by calling member functions of the interface that the objects have in common.

DEPR:

FIG. 3 shows a known browser environment 70 used to access information through the Internet. A client computer 20 uses a "browser" (e.g., Microsoft Corporation's Internet Explorer) to access documents and programs available on a remote computer called a server computer 74. The client computer 20 connects to the server computer over a telephone line 76 using a modem 78.

DEPR:

When used for browsing documents, the illustrated browser displays the document in a window 84 of the computer's display 30 allocated to the browser by the operating system. The illustrated window 84 comprises a frame 86, a document display area 88, and user interface controls 90. The browser displays the document within the document display area 88 of the window 84.

DEPR:

FIG. 6 shows a browser 170 (in this case, an Internet Explorer 4.0 browser) that is displayed if the user directly clicks on the teaser displayed in the viewer 140. The teaser image 152 and text 154, which are displayed in the viewer 140 of FIG. 5, may be also displayed in the browser 170. Of course, the teaser image and text need not be incorporated into the browser. Additional content 174 is also displayed to provide further information about the teaser 142.

DEPR:

The browser is a standard browser for displaying content, including hyperlinks. For example, a hyperlink 176 is shown as the underlined text "Southern Airlines." Thus, selecting hyperlink 176 allows a user to browse the Internet and display documents related to Southern Airlines, such as a Southern Airlines home page.

DEPR:

The system 220 may use smart or static servers to present channel guide information to a user. In either case, the server is a default server that provides the channel guide to the user upon selecting the channel guide button 150. The user accesses the server by selecting the channel guide button 150 or 184 in either the desktop window or the news window. The user may also access the server using a browser.

DEPR:

FIG. 9 provides further detail of how a channel guide 238 stores information in the user-preference storage 224. The channel guide 238 is preferably an HTML document hosting a control 240. The control is an object having an interface with functions for allowing the system to access (write and read) the user-preference storage 224. By convention, the control is designated by an object 242 having an interface 244, but hereinafter, controls are designated by a dashed circle embedded within a document, such as circle 246. The illustrated control is an ActiveX control, and may alternatively be one of many available controls, such as Java Applets, a Visual Basic scripts, or like objects. The control is embedded in the HTML channel guide by using special tags (e.g., Insert or Object tags).

DEPR:

FIG. 12 is a similar diagram to that of FIG. 11, with the addition of a special HTML document 314 containing a control 316. Content providers, such as content provider 318, can use the special HTML document 314 to display their content using different effects, such as effects 320, 322. The effects are easily updated and replaced on the client computer. The control 316 is used to parse information out of the content providers content files and display such information using the effects. The control 316 also stores information, such as a URL, into the user-preference storage 306 so that when a user selects or clicks on a teaser displayed in the viewer, the browser (FIG. 6) is automatically launched and the

URL is used to display the appropriate HTML document.

DEPR:

FIG. 15 shows a screen saver environment 350 which is controlled by an OLE container 352 hosting a control 354. The control 354 loads a ScreensaverWrap HTML file 356. ScreensaverWrap HTML has a control 358 embedded therein having the GetProviderInfo function. The control 358 reads user-preference storage 306 to obtain URLs saved in the user-preference storage based on selections made from the channel guide. The screen saver time slices screen saver documents, such as documents 360, 362, so that the documents are displayed sequentially. That is, if the content providers are a, b and c, then the screen saver first plays a screen saver document for a, then b, then c, and then repeats the process. Each screen saver document is played for the same amount of time, which is specified in a screen saver settings implemented in the container 352. The screen saver documents are HTML pages and are displayed full-screen. If a user clicks on a hyperlink in the screen saver 330, the browser is displayed to provide further information about the screen saver document.

DEPR:

FIG. 16 shows a screen saver environment 370 similar to the screen saver environment described in FIG. 15. However, a special HTML file 372 is provided which is capable of displaying plug-in effects, such as effects 374, 376, along with a document from a selected content provider. The special HTML file contains a control 380 having the same functionality as control 316 in FIG. 12. That is, it is used to store parameter strings in the user-preference storage 306, which are used if a user clicks on an element in the screen saver.

WEST Generate Collection

L21: Entry 16 of 29

File: USPT

Dec 26, 2000

DOCUMENT-IDENTIFIER: US 6167253 A

TITLE: Mobile data/message/electronic mail download system utilizing network-centric protocol such as Java

DRPR:

FIG. 27 illustrates a sample application using a browser as a front-end to a database;

DRPR:

FIG. 29 is an illustration of the software execution of a Java applet for the Java language; and

DEPR:

A more common architecture today involves forms on a web page to acquire input from the user, display information, and provide the familiar combination of text fields, radio buttons, scroll bars and list boxes. The most significant part of such architectures is the clickable button on the data entry form that says, in effect, "Yes! Send it now!" When the user clicks on that button, the Web browser executes a "submit" command, causing all of the data that has been entered into the form data fields to be sent back to the Web server. "Submit" is one of several examples of a standard CGI command, which invokes a program to process the data.

DEPR:

The Java paradigm also tends to ignore the presence of legacy applications. The Java paradigm is in the spirit of the Internet itself: open, free-wheeling, shareware kinds of applications will emerge and are likely to be quite popular. A development organization, however, needs something more organized and controlled. This does not make Java bad. On the contrary, even development organizations want to take advantage of distributed applets that can execute on a client platform. But a typical organization also wants an architecture that prevents access by the end user to other parts of the Web while an application is running. That is, typical organizations do not want end users to download some rogue applets that could cause some strange behavior.

DEPR:

On the other hand, the developer may wish to take advantage of applets and components on the Web to build an application; all of this could be stored within the organization's "database" of applications which could take the form of a private Web site. This ability to build applets is discussed below in detail.

DEPR:

Java is an object-oriented programming language with syntax similar to C and C++, only simpler. Because Java is an interpreted language, the typical C or C++ compile-link-load-test-debug cycle is reduced. Java development environments actually let the entire software-development life cycle take place within a Web browser.

DEPR:

None of these are the primary reason there is so much interest in Java. Rather, the main attraction is the fact that Java applications are completely portable. Write your code once and you never need to port or even recompile it. Rather than producing machine-specific instructions, the java compiler produces vendor-neutral bytecode. The Java runtime environment, or virtual machine, then translates the bytecode into actual machine-specific instructions. The Java virtual machine described below is installed on the user's machine, either as part of a Web browser or as part of the underlying operating system.

DEPR:

Like most object-oriented programs, all variables and functionality in a Java

program are contained within objects, and objects are defined via classes. Unlike traditional structured languages, there are no stand-alone functions or subroutines. A class contains both variables (data structures) and the methods (functions) that operate on these variables.

DEPR:

FIG. 24 is an illustration of the software implementation architecture for the Java language. As illustrated in FIG. 24, programs or applets 410 written in Java are compiled to an intermediary form called a byte code 412, and this byte code 412 is translated by a Java interpreter, e.g., 416, 420, 424, also called a virtual machine, into code that is used by the Web browser, e.g., 414, 418, 422, to perform system functions. While the interpreted code does run fairly fast, the interpreter 416, 420, 424 must be designed to run on a particular machine, and the browser 414, 418, 422 must be tailored to a specific machine.

DEPR:

As illustrated in FIG. 25, when surrounded with an ActiveX-based browser 426, the Java interpreter 428 need not be specific to a particular system to translate Byte codes 412 compiled via Java applet 410. As the latest implementation of Microsoft's OLE technology, ActiveX is compatible with other object-oriented/scripting tools 436, via ActiveX scripting 434, developed by Microsoft, and therefore, lets programmers use tools with which they are already familiar.

DEPR:

The latest additions to the language adds features that will coordinate the position and status of shared objects and avatars--cyberspace stand-ins for actual people. The Living Worlds standard 448, 450 also deals with non-VRML elements and interactions among objects. With its aid programmers will be able to identify and integrate run-time interaction capabilities that are implemented outside VRML and its scripts, e.g., 454-470, to effect information exchange between objects in 3-D environment, and to define the rules and the roles for interaction. The standard will also handle some system security issues. The upshot will be to give avatars greater freedom to both share and interact in a virtual reality environment.

DEPR:

Early users of Java focused on applets, or mini-applications, downloaded as part of a Web page. To understand how this works, the components in a Web page containing a java applet are to be examined. The Web consists of browsers and servers connected over TCP/IP networks. When a browser is instructed to fetch a particular URL, it starts by formatting a request to get the specified resource from the specified host. The typical home page returned to the browser is formatted using Hypertext Markup Language (HTML). While HTML does not contain any native graphics formats, many home pages include graphics by using the img tag. When a browser encounters an img tag, it sends another request to the server to download the named image.

DEPR:

In the same fashion, when a Java-enabled browser encounters an applet tag, it sends another request to the server to download the named Java applet. The Java code for a simple applet that draws a sine wave is illustrated below:

DEPR:

After compiling the file Wave.java, the compiled version for embedding a Java applet in a Web page is shown below:

DEPR:

Java applets and browser plug-ins are both ways to add new functionality to browsers that support them. There are major differences, however, between the two technologies. For starters, Java applets are automatically downloaded to a Java-enabled browser whenever you visit a Web page containing them. To take advantage of a Web page that uses a "plug-in" application, you must first install the plug-in and configure it to work with your browser. Secondly, since a plug-in is compiled code, it is inherently not cross-platform or standard. For instance, there is no standard way for a bar graph plug-in from vendor X to send its output to a spreadsheet plug-in from vendor Y on the same Web page. One of Java's strengths is that it can be easily extended to handle as-yet unspecified file formats. When the browser encounters an unknown resource, it simply requests that the appropriate protocol handler from the server.

DEPR:

Layer 3: Classloader. A Java applet may call a number of different classes, including built-in classes loaded from the local file system, classes from the same server, or classes loaded from elsewhere in the network. For further security, each imported class (one loaded from the network) executes within its own separate name space. There is a single name space for all classes loaded from the local file system, since the security of these can be locally maintained. When a class references another class, it first looks for it in the name space of classes loaded from the local file system (the built-in classes) and then in the name space of the referencing class. There is thus no way any Java code loaded over the network can "spoof" a built-in class or a class loaded from another network site. In a similar fashion, the built-in classes can never accidentally reference classes in imported name spaces.

DEPR:

Many of today's large, enterprise-wide distributed systems include one or more relational databases such as those sold by Oracle, Sybase, and Informix. Web developers discovered early on the simplicity of using a browser as a front-end to a database.

DEPR:

FIG. 27 illustrates a sample application using a browser as a front-end to a database. The developer of an electronic time card system that uses a relational database as the underlying storage model would much rather use an HTML form as the data-entry screen than port the data-entry application to multiple platforms. An employee enters data into the HTML form such as employee number and hours worked, and clicks a "submit" button. The data is buffered and returned to the Web server, which then uses standard CGI to launch a separate application program that validates the data and inserts the records into the database.

DEPR:

As illustrated in FIG. 27, the Web server process 474 of the Web machine 472 transmits at (1) the time and card form via network 476 to browser 478. Browser 478 loads the time card form 480 and transmits same to the Web server machine 472 and Web server process 474 and the user inputs data at (2). Web server process 474 submits the form to the time card script 482 at (3), and the script parses the form and returns any error messages to the Web server process 474 at (4). The script 482 then inputs the time card data to the standard DBMS server machine 484 and the DBMS server process 486 running therein at (5). The DBMS server machine 484 inputs the data and returns acknowledgement to the time card script 482 at (6). The time card script 482 returns acknowledgement to the Web server process 474 at (7).

DEPR:

FIG. 28 shows how the Java JDBC Application Programmer Interface (API) allows the time card application to communicate directly with the DBMS, eliminating many of the problems of a traditional Web-based system. In this case, the browser loads the time card applet, the user enters the data, and the applet validates the fields locally before sending the data directly to the DBMS via JDBC.

DEPR:

As illustrated in FIG. 28, the Web server process 490 of the Web machine 488 transmits at (1) the time and card form via network 492 to browser 494. Browser 494 loads the Java time card applet 498 and the user enters data, and the applet validates the form fields locally at (2). Data is then sent directly by JDBC 496 to the DBMS server machine 500 and the DBMS server process 502 via network 492 at (3).

DEPR:

FIG. 29 is an illustration of the software execution of a Java applet for the Java language. In FIG. 29, Java source programs or applets 504 are compiled at 506 to an intermediary form called a bytecode or Java bytecode 508, and this byte code 508 is transmitted to Bytecode loader 512 via network 510 where it is loaded into the machine. The compiled applet is then verified by Bytecode verifier 514 that the Bytecode corresponds to Java code and does not contain any viruses.

DEPR:

Supports the creation of aglets (agent-applet). Supports the notion of an Agent

Transfer Protocol (to handle shipping and transferring aglets), object passing, autonomous asynchronous execution, and disconnected operations. J-APPI specification allows for interfacing aglets and the environment in which they are allowed to execute. J-APPI supports the creation of aglets, dispatching them to remote locations, retracting them from remote locations, activating and deactivating aglets, cloning them and disposing of them. Aglets execute in an aglet context and they can have an itinerary as to where they intend to migrate and execute.

DETL:

```
import java.awt.*; import java.applet.*;
public class Wave extends Applet { int n = 1 public void paint (Graphics g) {
double y=0.0. oy=0.0; for (int x=0; x < size ( ).width; oy=y, y=0, x++) { for (int
j=o; j
```

DETL:

This simple applet example draws a sine wave.

<applet codebase="classes" code="Wave.class" width=600 height=100>

WEST **Generate Collection**

L21: Entry 21 of 29

File: USPT

Jul 4, 2000

DOCUMENT-IDENTIFIER: US 6085224 A

TITLE: Method and system for responding to hidden data and programs in a datastream

ABPL:

A system and method for detecting trigger events in datastreams received over a computer communication network is disclosed. The system includes an interceptor for intercepting datastreams from a computer network intended for an application program; a scanner for scanning the intercepted datastream for trigger events, such as cookie data, script commands, and applet programs; and an event response generator for processing detected trigger events. Configuration data is used to identify a response for trigger events such as disabling script commands or programs and deleting or modifying cookie data. The event indicators and an action menu are generated by the event response generator and delivered with the processed datastream to the application program. The application program displays the event indicators so the user is made aware of the trigger events and the action menu allows a user to respond to the detected trigger events. In the preferred implementation, the user may respond by obtaining information about the site which transmitted the datastream having the trigger events and then send e-mail to the administrator of the site. Other actions include modifying the configuration data so subsequent datastreams with the trigger event is passed by the system. Outbound messages from the application program are also intercepted and scanned for trigger events. In the preferred implementation, the configuration data are exchanged between the system and the application program so the user may modify the operation of the system. The configuration data are deleted from the outbound datastream before it is transmitted in the preferred implementation. The system and method of the present invention allow a user to view detected trigger events which otherwise would occur without the user's knowledge and provides the user with sufficient information so the user can make an informed decision as to whether to accept trigger events in a datastream from another site.

BSPR:

A popular datastream protocol over the Internet is the Hypertext Transport protocol (HTTP). This protocol is used to transfer and display information, usually in a graphic format, from one computer to another. The files containing the information to be displayed are usually written in the Hypertext Markup Language (HTML). The HTML language includes commands which are executed by a program at the receiving computer. The files also include identifiers for files which include information to be displayed. These file identifiers are typically known as Universal Resource Locators (URL). The program at the receiving computer which displays information received from another computer in an HTML file or which returns user information to the program which sent the HTML file is commonly known as a browser. These browsers are typically referred to as client programs and the computers sending HTML files and the files corresponding to URLs within the HTML files are known as servers. The portion of the Internet which communicates in the HTTP protocol is usually referred to as the World Wide Web (WWW).

BSPR:

In a response to a request for a page sent by a browser, a server sends multiple HTML files which comprise the page in messages implemented in the HTTP protocol. When the HTML file or files are received by the computer executing the browser, each communication stack layer performs its function until a datastream containing an HTTP header and corresponding data segment is presented to the browser. One portion of the browser verifies that the information and the HTTP header have been accurately delivered to the application program. The browser then displays the data delivered in the HTML files received from the server. Because the TCP/IP protocol used for the Internet is a packet communication protocol, several messages are probably required before a complete file is available for display. Besides graphical data, the HTML file also contains data and/or commands which may

not be displayed at the browser. This "hidden" data and/or commands may be used to cause the computer executing the browser to store information or execute programs without the user's knowledge of the existence or purpose of the information or program.

BSPR:

One known data field which may be included in the HTTP header of an HTML file is a "cookie" data field. A cookie is an HTTP protocol header document element which may be used to provide multiple data elements to the browser. In response to receiving an HTML file with a cookie, the browser may store the cookie data elements in a "cookies.txt" file which is usually kept in the root directory for the browser. Once cookie data are sent to the browser computer, the server expects the cookie data to be returned in the HTTP header of subsequent messages sent from the browser to the server. The inclusion of the cookie data in the HTTP header of messages from the browser is done without the user's awareness. In this manner, the operator of the server may identify repeat visitors to the server site. Other known methods of passing cookie data to a client program include using a Javascript data object or a Javascript program that accesses the "cookies.txt" file stored at the client computer. While the storage of a cookie file may appear harmless, it is nevertheless the unauthorized storage of data on another's computer and the file may be used for tracking the user and his or her requests for information from the server site without the user's knowledge or permission.

BSPR:

Some known programs may be used to scan HTTP headers of HTML files and requests before the files are processed by a user's browser or the request from a browser is sent to the communication stack for transmission. These programs may be used to detect cookie data in incoming files and outgoing requests. These programs allow a user to activate a function which notifies the user of cookie data in HTTP headers of incoming HTML files. The user may also activate a function of the program to delete the cookie data from the HTTP header of incoming file so it is not passed to the browser program and stored in the cache memory for the browser. If the cookie data is stored in the cache memory for the browser, the browser incorporates the cookie data in the HTTP header of outgoing HTML GET or PUT requests from the browser. These previously known programs may be used to notify the user of cookie data in the HTTP headers of the outgoing requests and to delete the cookie data from the HTTP headers independently of notification of cookie data for incoming files. These programs are separate from a user's browser and thus, may be advantageously added to a user's system without modifying the executable code for implementing the browser program. While these previously known programs may be used to selectively notify a user of the presence of cookie data in an HTTP header or to delete cookie data from an HTTP header, these programs do not detect other hidden data which a user may want to know is being passed to the user's browser or want to delete from an HTML file or request.

BSPR:

Recently, powerful interpretive languages have been developed which may be executed in a browser. Known interpretive languages are JAVA developed by Sun Microsystems, Javascript developed by Netscape Communications Corporation, and Visual Basic Script developed by Microsoft Corporation. Because each one of these languages are interpreted, a program written in one of these languages does not need to be compiled with prior knowledge of the computer on which it will execute. Instead, the interpreter executes within the application space for the application program, such as the browser, and this interpreter executes statements received in a file containing the interpretative language statements. Files containing interpretive language statements are known as applets. While applets have a number of beneficial purposes, they may also cause problems. For example, a JAVA applet may be imbedded in an HTML file, sent to a user's computer and executed by an interpreter in the browser without the user's knowledge. Such programs may be used to gain unauthorized access to resources or data on the user's computer. Additionally, these interpretive language programs may include cookie commands that identify tracking data as discussed above. These cookie commands are part of the data segment of a datastream for a browser and not part of the HTTP header. As a result, these cookie commands are not detected by the programs that may be used to detect and delete cookie data from HTTP headers.

BSPR:

To address the need to detect interpretive language programs and cookie commands data segments of datastreams, some known browsers have been modified to include a

function which a user may activate to prevent the execution of interpretive language programs and cookie commands. Typically, the browser is modified so the portion of the browser program that passes an interpretive language program or cookie command to an interpreter for execution, checks a switch which may be set by a user, to determine whether passing programs and commands to the interpreter is enabled. While these modified browsers disable the execution of interpretive programs and cookie commands, they do not notify a user that an interpretive program or cookie command was detected. Thus, users are unaware of those server sites that attempt to send interpretive programs and cookie commands to the user's browser and, as a result, the user may deactivate the interpretive program and cookie command disabling function of the browser. Thereafter, the user may request an HTML file from a server previously visited and receive an interpretive program or cookie command that now executes on the user's computer. If the user had known the server site was sending interpretive programs or cookie commands, the user may have chosen not to request files from the server.

BSPR:

What is needed is a program which detects programs or cookie commands embedded within a datastream received from another computer and which notifies the user of the interpretative language program or cookie command so the user may be aware that the server is sending interpretive programs or cookie commands. What is needed is a program which notifies the user of detected interpretive programs and cookie commands without modifying the browser program. What is needed is a way to restrict access to resources or data on a computer when the computer is in communication with another computer.

BSPR:

In a preferred embodiment of the present invention, the interceptor is a program which overloads a portion of the socket program that communicates directly with the transport layer. Thus, the interceptor can receive a datastream from the transport layer prior to its delivery to an application program and can receive a datastream to be sent to another computer before it is received by the transport layer for transmission to the other computer. Alternatively, the interceptor may execute in the application space with an application program, usually a browser, and intercept datastreams to and from the application program before they are processed by the application program or sent to the transport layer, respectively.

BSPR:

Preferably, the action menu presented by the event response generator provides a user with options to (1) generate an electronic (E-mail) message to the system administrator who operates the server site which caused the trigger event, (2) terminate the communication session with the server site which caused the trigger event, or (3) modify configuration data corresponding to the server site which is used to process detected trigger events. Configuration data are a user's identification of the types of actions the inventive system performs for the hidden data elements or commands detected in datastreams. For example, if a user needs to allow the downloading of applet files and the commands to execute them in order to view a desired resource from a server site, then the user may permit such program downloading and command executions. The event response generator also generates and stores a log of each trigger event. The log records for the trigger events preferably include the time of the trigger event, identification of the source of the trigger event, the type of trigger event, and any file which may have been received as a result of the trigger event. The user may view this event log to ascertain what various server sites are providing in datastreams to an application program which otherwise would be undetected.

BSPR:

In the method of the present invention, an inbound datastream is intercepted prior to it being delivered to an application program. The datastream is then scanned for trigger events and processed according to the configuration data for the server site. If the configuration data indicates that script programs are to be disabled, for example, any script program in the datastream is disabled so it cannot be executed by the browser. The detected trigger events are then logged and a data envelope containing event indicators and an action menu is generated and coupled to the processed datastream. The browser program then displays the processed datastream, the event indicators and the action menu. The user may select actions in the action menu to view detected trigger events or initiate control actions. For example, a user may view a disabled script program and

conclude its execution is acceptable to the user. The user may then modify the configuration data to allow script program execution and then cause the browser to request the page from the server again. Upon receipt of the requested page, the system of the present invention does not disable the script program and it will execute when the datastream is passed to the browser.

DEPR:

System 10 may execute in the process space of an application program, such as a browser, or the process space of a communication program, such as a WINSOCK program. A browser typically communicates datastreams with servers using the HTTP protocol over an open network such as the Internet. A communication program establishes a communication session between the application program and communication stack 12. The communication program communicates with communication stack 12 so system 10 is coupled to a network. As explained above, communication stack 12 includes transport layer 18, network layer 20, data link layer 22, and hardware component 26. When system 10 executes in the process space of a communication program, it overloads a portion of the communication socket program to "hook" or intercept datastreams being communicated to and from the application program. A WINSOCK communication program used in the preferred embodiment of the present invention is the WINSOCK COMPONENT ARCHITECTURE program available from Stardust Technologies, Inc. of Campbell, Calif. When system 10 executes in the process space of the application program, it intercepts and processes datastreams from the communication program, such as a WINSOCK program, before determining whether to provide them to the application program. Likewise, it intercepts and processes datastreams from the application program before determining whether to provide them to the communication program.

DEPR:

Preferably, system 10 is a program implemented in the C computer programming language and operates on a personal computer (PC) or the like. At a minimum, the computer executing system 10 should have an Intel 80486 processor or equivalent, 16 MB of RAM, 500 MB of hard disk space, a VGA monitor, keyboard and a mouse. Preferably, the program implementing system 10 operates on a computer using a Windows operating environment. For 16 bit processors, the program preferably operates in the Windows 3.1 or Windows for Work Groups 3.11 environments and is preferably compatible to operate in the process space of a sixteen (16) bit WINSOCK program or as a plug-in application to the Netscape Navigator 3.0 browser application. For 32 bit processors, the program implementing system 10 preferably operates in the Windows 95 or NT environments and preferably operates in the process space of a thirty-two (32) bit WINSOCK or as a plug-in application with the Netscape Navigator 3.0 browser or as an Active-X control program for the Microsoft Explorer 3.0 browser.

DEPR:

In the preferred implementation, the most significant digit in the preferred 10 digit action map data defines whether hidden data from a server site is displayed in the browser. In response to this digit having a value of `0`, event response generator 34 deletes cookie values in an HTTP header or in embedded script "document.cookie" commands of a datastream received from the server site, sets the cache value in the HTTP header so data is not stored in the cache of the user's computer, and deletes at browser initiation or termination cookie data in the "cookies.txt" file which were received from unauthorized servers. In response to this digit having a value `1`, event response generator 34 deletes the expiration dates for cookie data in HTTP header so the cookie values from the server site may be received, sets the cache value in the HTTP header so data is not stored in the cache of the user's computer, and deletes at browser initiation or termination cookie data in the "cookies.txt" file which were received from unauthorized servers. The second digit of the preferred action map data defines whether data from a server is stored on the user's disk drive. In response to this digit having a value of `0`, event response generator 34 deletes cookie values in an HTTP header for a datastream received from the server site, sets the cache value in the HTTP header so data is not stored in the cache of the user's computer, and deletes at browser initiation or termination cookie data in the "cookies.txt" file which were received from unauthorized servers. In response to this digit having a value `1`, event response generator 34 allows cookie values with valid expiration dates to be stored in the "cookies.txt" file and datastreams from a server may be stored in the cache of the user's computer and on the user's disk drive. The third digit of the preferred action map data defines whether data from a server stored on a user's disk drive may be returned to the server. In response to this digit having

a value of `0`, event response generator 34 deletes all cookie values in an HTTP header for outgoing datastreams to the server site. In response to this digit having a value `1`, event response generator 34 allows cookie values to be returned to the server site which sent them in a previous datastream. The fourth digit of the preferred action map data defines whether a server can request data from another site for display by a user's browser. In response to this digit

having a value of `0`, event response generator 34 deletes all HTML "....." statements where the URL identifies a server other than the one with which communication is active. In response to this digit having a value `1`, event response generator 34 allows GET requests for URLs from other server sites to be sent to the user's computer for display. The fifth digit of the preferred action map data defines whether a server can request data from another site be sent to the user's browser for storage. In response to this digit having a value of `0`, event response generator 34 deletes all HTML "Set Cookie" statements where the URL in the datastream identifies a server other than the one with which communication is active and no cache storage is allowed for datastreams having a server address different from the one with which communication is currently active. In response to this digit having a value `1`, event response generator 34 allows cookies from other server sites to be stored on the user's computer.

DEPR:

Continuing with the description of the preferred implementation, the sixth digit in the preferred 10 digit action map data defines whether programs from a server may be executed in the display of a user's browser. In response to this digit having a value of `0`, event response generator 34 disables HTML SCRIPT and APPLET tags as well as Javascript "document.applet" commands. In response to this digit having a value `1`, event response generator 34 allows HTML SCRIPT and APPLET tags as well as Javascript applets to execute in the display of a user's browser. The seventh digit of the preferred action map data defines whether programs from a server may execute on a user's computer. In response to this digit having a value of `0`, event response generator 34 disables HTML OBJECT and EMBED tags as well as Javascript "document.embed" commands. In response to this digit having a value `1`, event response generator 34 allows HTML OBJECT and EMBED tags as well as Javascript embedded commands to execute on the user's computer. The eighth digit of the preferred action map data defines whether browser and e-mail user information data may be sent to a server. In response to this digit having a value of `0`, event response generator 34 deletes "User-Agent" and "From:" fields from HTTP headers of outbound datastreams. In response to this digit having a value `1`, event response generator 34 allows HTTP headers having "User-Agent" and "From:" fields to be transmitted in outbound datastreams. The ninth digit of the preferred action map data defines whether page updates from a server may be received by a user's browser. In response to

DEPR:

this digit having a value of `0`, event response generator 34 deletes all "Connection:keep-alive" and "refresh" statements from HTTP headers in inbound datastreams. In response to this digit having a value `1`, event response generator 34 allows "Connection:keep-alive" and "refresh" statements to remain in HTTP headers in inbound datastreams so they are processed by a user's browser. The tenth digit of the preferred action map data defines whether disk I/O is active during network communication. In response to this digit having a value of `0`, event response generator 34 allows disk I/O to remain active during an active TCP/IP socket connection. In response to this digit having a value `1`, event response generator 34 disables disk I/O whenever a TCP/IP socket connection is active. Although these are the preferred actions and their preferred implementations in the present invention, other actions and implementations may be used without departing from the principles of the present invention.

DEPR:

Thus, the event configuration data defines the actions to be performed for each type of trigger event. When event response generator receives an action map from the user's browser, the event configuration data is modified to conform to the action map. In this manner, the user need only specify actions and the inventive system correlates the specified actions to the trigger events for which scanner 32 scans. In the preferred implementation, the most significant digit in the preferred 12 digit map data defines whether cookie values in an HTTP header for a data message received from another computer are modified or stored in a

"cookies.txt" file. As shown in the table, a zero value for the digit indicates that no modifications are made to cookie values in the header, a "1" value for the digit indicates that a user may modify the value before it is returned to the computer which sent the message with the cookie data in the header, and the value "2" causes the file ("cookies.txt") in which cookie data are normally stored to be deleted upon initiation of the application program or upon termination of the application program. For the second map data digit, the value zero permits refresh files to be received and displayed by the application program and the value "1" deletes refresh file requests from outbound datastreams. Refresh files are typically HTML files sent by a server to update an area within a previously transmitted page. The third map data digit determines whether repeat images from the server are displayed by the application program. A value of zero permits the repeat data files to be received and displayed, a value of "1" preferably causes the TCP/IP socket to close after a Web page has been downloaded so repeat data files are not received. The fourth map digit defines whether MIME encoded files are decoded and displayed by the application program. A value of zero permits all MIME encoded files to be decoded and used by the application file, a value of "1" permits those MIME files containing text and image data only to be received and decoded by the application program, and the value "2" permits those MIME files containing text only to be received and decoded. The fifth map data digit either allows or disables execution of script commands received in a datastream. A value of zero for this digit permits the application program to receive and execute script commands while a value of "1" causes the scanner to disable script commands. Preferably, script commands are disabled by placing a comment character before the script command so the detected script command may be displayed by the application program for the user. In the preferred implementation, detected script commands are in Javascript or Visual Basic Script (VBS) languages. The sixth map data digit determines whether cookie data in script commands are permitted. That is, cookie data may be defined in script commands to avoid being detected in HTTP headers. If this map digit is zero, the script cookie data are received and used by the application program. If the map data value is "1", detected cookie data in script commands are disabled and displayed by the application program for the user so the user may determine whether receipt of the cookie data in the script command is allowed. The seventh digit in the map data determines whether a FORM "submit" script command will be provided to the application program. A digit value of zero allows FORM "submit" script commands to be received and executed by the application program and the value of one causes the scanner to disable the FORM "submit" command so it may be displayed for the user by the application program without execution. The eighth digit in the map data determines whether a script program may execute a plug-in program using an "embed" command. A value of zero permits script programs to execute plug-in programs with "embed" commands and a value of "1" causes event response generator 34 to disable "embed" commands for plug-in programs in script programs. Preferably, script commands are disabled by either placing a comment character in front of the command which invokes the program or by inserting a return statement as the first statement in the imbedded program function so the body of the imbedded program function is not executed. The ninth digit of the map data determines whether an applet program is received and executed. When this digit is a zero, applet programs may be received and executed by the application program and when this digit is a "1", event response generator 34 disables the applet program. Again, the preferred implementation disables the applet program by either placing a comment character before the command invoking the program or placing a return statement as the first statement in the applet program. The tenth digit in the map data determines whether imbedded programs for a plug-in application are received and executed by the application program. A value of zero for this digit permits imbedded programs to be received and executed by the plug-in application and a value of "1" disables the imbedded programs for the plug-in application. Again, the preferred implementation disables imbedded programs or plug-in applications in the manner discussed with respect to applet programs. The eleventh map data digit determines whether data objects are passed and used by the application program. The map value of zero permits data objects to be received and used by the application program and a value of "1" disables data objects so a detected object may be displayed by the application program for the user. The data object is preferably disabled by placing a comment character in front of the command which uses the data object or by detecting the program portion component of the data object and disabling it by putting a return statement as the first statement in the program portion of the object. In the preferred implementation, data objects are typically data objects written in the Active X language. The twelfth digit of the map data determines whether disk I/O is disabled. When the value of this digit is at zero, disk I/O is disabled and

files and data are written to a RAM area. When this digit is a one, files and data may be stored on a disk drive.

DEPR:

In the preferred implementation, the action map data is assigned by a "Cookie:iv=map data" statement. The "cookie:iv" portion of the statement specially defines the cookie data as being generated by system 10 so system 10 can distinguish the cookie data used to control system 10 from cookie data sent from a server site or the application program. A diagrammatic depiction of the preferred processing of an HTML file by system 10 is shown in FIG. 2. As shown there, a server sends an HTML file having cookie data in the HTTP header (Step A). This file is intercepted by system 10 and scanned to detect trigger events in accordance with event configuration data stored in configuration file 38 which corresponds to the server. Scanner 32 determines which configuration data to use by extracting the server address in the HTTP header and searching configuration file 38 for an entry which corresponds to this server address. If one is located, the information in the corresponding event configuration data is used by scanner 32 to detect trigger events and action map data are used by event response generator 34 to generate appropriate responses for the trigger events detected by scanner 32. If no corresponding configuration data have been stored for the server site, a default action map value is used to set default event configuration data. In the preferred implementation, the default action map value is "iv=0000000001", although other values may be used. The action map data are stored in the HTTP header of the processed datastream (Step B). Event response generator 34 also couples an HTML file to one end of the datastream, preferably, at the trailing end of the intercepted datastream (Step C). A preferred HTML form used to accept user input for modification of the preferred action map data is shown in FIG. 3. Preferably, other HTML may be included in the HTML file to provide the user with event indicators and other action options such as server administrator inquiries and responsive e-mail generation as discussed below. In the preferred HTML

for the preferred action map, the "reset page" option permits a user to request the datastream for the downloaded page from the server again so it may be processed with the new event configuration data before it is delivered and displayed by the user's browser. Coupled to the end of the preferred HTML file generated by event response generator 34 is a Javascript program which performs the actions presented in the action menu. A preferred implementation of a Javascript program for modifying the action map data is shown in FIG. 4.

DEPR:

The processed datastream with the specially defined cookie field and trailing HTML file is passed to the browser for display (Step D, FIG. 2). The HTML forms of the trailing HTML file are displayed to reveal event indicators and the action menu to the user. The HTML form for the event indicators may accept activation by the user to display the detected trigger event in the HTML file for the processed datastream. For example, if an event detector indicates an applet program has been detected, a user may "click" on the indicator so the HTML source code for the detected applet program is displayed to the user. The displayed program contains the comment character or return statement used to disable the program. In this manner, a user may view detected trigger events and determine whether the detected trigger event is acceptable to the user.

DEPR:

Because trigger events are detected and processed without user intervention, system 10 can process an entire page from a server and then deliver the processed page to the application program with the detected trigger events disabled. Previously known programs that detected or deleted cookie data from HTTP headers were unable to detect and disable interpretive programs and embedded cookie commands. Previously known browsers that disabled interpretive programs and embedded cookie commands did not notify the user of disabled interpretive programs and embedded cookie commands. The system of the present invention provides a program which may execute in the application space with a browser or overload part of the communication socket

program to detect and delete cookie data from HTTP headers as well as notify a user of interpretive programs and embedded cookie commands disabled by the system. With the system of the present invention, a user may view a complete page with the detected trigger events disabled, be notified of the detected trigger events and then determine whether detected trigger events are acceptable for subsequent datastreams.

DEPR:

Examples of embedded commands include any commands which activate or execute a program or applet. Programs which may be activated by an embedded command include those written in the JAVA script or Visual Basic Script languages. Also, JAVA applets, Navigator plug in applications, and Microsoft Active-X control applications are programs that may be activated by embedded commands. For example, the HTML tag "APPLET" may be used to invoke a JAVA applet, the "EMBED" HTML tag may be used to invoke Netscape plug-in of applications, and the "SCRIPT" HTML tag may be used to invoke Javascript or Visual Basic Script programs. Thus, identification of these HTML tags in a file received from a server site is a detection of a trigger event for an embedded command. Also, Javascript programs received from a server site are scanned to determine if a "document.applets" string or "document.embeds" string is contained in the Javascript program. These two examples of JAVA script language statements are used to invoke a JAVA applet or a Netscape plug-in application, respectively. Again, these commands which activate programs or applets are merely examples of the types of commands which may be detected by scanner 32 in a scan of an HTML file or downloaded program or applet file.

DEPR:

An example of an unidentified file request in an incoming datastream is "IMG SRC="browser is communicating.

DEPR:

The preferred method implemented by system 10 for inbound data messages is shown in FIG. 5. The method begins by intercepting a datastream before it is received by the application program in the application space shared with system 10 and the event configuration data are retrieved from configuration file 38 (block 50). The datastream is scanned for trigger events defined by event configuration data corresponding to the server which sent the datastream (block 54). The reader should appreciate that the term datastream includes the data components of a data communication between computers as well as the header information for the data segment. If the scan of the datastream indicates a trigger event is present (block 56), the process writes the detected trigger events to a log file (Block 60). Regardless of whether trigger events were detected or not, action map data corresponding to the server site which generated the inbound datastream is placed in the header of the HTML file containing the processed datastream (Block 68). If action map data corresponding to the server site is not available, the default value for action map data is used instead. The process then determines whether the action map data indicate a response should be generated for a detected trigger event (Block 70). If a response is indicated by the action map data, the response is executed (Block 72) and an event indicator for the processed trigger event is included in a data envelope (Block 74). For example, if an applet program is detected, the executed response results in the applet program being disabled and an HTML statement to display an event indicator is included in the data envelope for display by the application program. Preferably, the data envelope is the HTML file coupled to one end of the datastream as discussed above. This process is repeated for each protected trigger event (Block 76) until all of the detected trigger events have been processed. The process then generates an action menu providing selections for the detected trigger events (Block 78) and the action menu is included in the data envelope (Block 80). The data envelope is then coupled to one end of the datastream (Block 82) and the action map

·data used to process the datastream is placed in the header of the datastream (Block 84). The processed datastream and data envelope coupled to the datastream are then provided to the application program (Block 86).

DEPR:

The datastream and coupled data envelope are then displayed by the application program and the user may view the detected events. A user may then select an action in the action menu. In the preferred implementation, the actions in the action menu are performed by a script program invoked from the action menu. The actions available to a user in the preferred implementation include obtaining more information about the server site which sent the detected trigger event or modification of the action map data so a trigger event may be used or executed. For example, to obtain more information about the server site, the user may send a WHOIS query to the domain name registration authority. This query is sent in the preferred embodiment by establishing a HTTP session, or alternatively a TELNET communications session, for transmission of the query to the domain name registration authority. The response to the query identifies the owner of the server site, its administrator, company name, and e-mail address. The user may now activate a selection in the action menu to send the identified administrator an e-mail regarding the detected trigger event. In the preferred implementation, a default message is provided for transmission to the administrator. In another aspect of the preferred implementation, the user may activate a selection in the action menu to send a FINGER query to the server site to obtain information regarding the administrator of the site. This information may also be used to send an e-mail to the administrator regarding the detected trigger event. After viewing the detected trigger event, a user may determine that the detected trigger event is acceptable to the user. In this case, the user selects an action from the action menu which modifies a value in the action map data so that subsequent transmissions permit the trigger event to be received and used or executed.

DEPR:

In operation, a user installs system 10 of the present invention on a computer as it is activated with an application program. Preferably, the installation associates system 10 with a user's browser so system 10 is activated whenever a user activates the browser. Thereafter, when the user activates the browser, system 10 operates to intercept outbound datastreams before they are processed by communication stack 12 and intercept incoming datastreams before they are processed by the user's browser. These datastreams are scanned to detect trigger events and detected trigger events are logged. If a response for a detected trigger event is identified by configuration data, the response is generated. Event indicators and an action menu are then generated and an HTML file containing event indicators and the action menu is coupled to the datastream. The action map data is also stored in the header of the datastream. The event indicators and action menu are presented and any action selected by the user is executed. If no trigger event is found in a datastream, an action menu is coupled to the datastream for delivery to the application program so the user can modify the action map data, if desired.

DETL:

MAP DATA DIGIT POSITION ACTION VALUES

1

temporary storage '0' = no temporary storage of displayed data '1' = temporary storage of displayed data
2 disk storage '0' = no disk storage of server data '1' = allow disk storage of server data
3 hidden data return '0' = no return of server hidden data '1' = allow return of server hidden data
4 relayed server data '0' = no relayed server data displayed '1' = relayed server data displayed
5 storage of relayed server data '0' = no relayed server data stored '1' = relayed server data stored

6 server programs in display window `0` = no program execution in display window `1` = program execution in display window 7 server programs on user's computer `0` = no program execution on user's computer `1` = program execution on user's computer 8 e-mail user information `0` = no browser or e-mail user data to server `1` = browser or e-mail user data to server allowed 9 page updates `0` = no page updates `1` = page updates allowed 10 disk access `0` = disk access disabled `1` = disk access allowed

DETL:

MAP DATA DIGIT POSITION RESTRICTION VALUES

1

cookie.sub.-- modifications `0` = no modifications `1` = modify name = value statement or expiration date for cookie value `2` = delete cookies.txt at start-up/shut down 2 refresh, `0` = allow always `1` = delete always 3 keep.sub.-- alive, `0` = allow always `1` = delete always 4 mime.sub.-- type, `0` = allow all `1` = text and image only `2` = text only 5 script, `0` = allow always (Java Script or VBS) `1` = disable always 6 script.sub.-- cookie, `0` = allow always `1` = disable always 7 script.sub.-- submit, `0` = allow always `1` = disable always 8 script.sub.-- embed, `0` = allow always `1` = disable always 9 applet, `0` = allow always (Java) `1` = disable always 10 embed, `0` = allow always (Plug In) `1` = disable always 11 object; `0` = allow always (Active X) `1` = disable always 12 Virtual RAM Drive `0` = disable `1` = enable

CLPR:

15. The method of claim 9 wherein the intercepting step intercepts the datastream in an application process space for a browser program.

CLPV:

a script language program for implementing said action menu.

WEST **Generate Collection**

L21: Entry 22 of 29

File: USPT

Jul 4, 2000

DOCUMENT-IDENTIFIER: US 6083276 A

TITLE: Creating and configuring component-based applications using a text-based descriptive attribute grammar

BSPR:

The present invention relates generally to object-oriented programming, and more particularly, to a method and system for creating and configuring a component-based, object-oriented program using a text-based descriptive attribute grammar.

BSPR:

Although the above systems offer solutions to a variety of development problems, none, however, allow applications to be effectively created and configured using a text-based application description language based on an SGML-compliant syntax. For example, integrated development environments, component-based frameworks, and most visual authoring tools output object-code binary files which are interpreted and executed by the user's operating system or the given language's "virtual machine." Java "applet" builders, for instance, typically generate Java source or object code directly. Such binary code files are not text-based and thus are neither editable nor accessible to text-oriented tools.

BSPR:

Although interpreted scripting languages such as JavaScript, Hypertalk, and Lingo can be edited in a generic text editor, their syntax is procedural rather than declarative. As a result, the semantic content of these script files can only be productively inspected and analyzed by the specific tools and editors dedicated to that particular language. In short, while there are a wealth of powerful component-oriented application frameworks and authoring tools, none of them are based on a standardized, text-based descriptive grammar or realize the advantages which accompany being based on such a grammar.

BSPR:

Analogously, numerous efforts have been made to extend the ability of SGML and HTML to specify more sophisticated formatting instructions. For example, HTML's pending versions include numerous processing and formatting instructions to assist developers in creating "multimedia" documents that provide basic interactivity through external "scripting" languages that control "black box" components such as Java applets and/or ActiveX components. Originally, HTML was intended as a purely descriptive syntax that would be transparently portable and easy to understand by both technical and non-technical publishers. The newly evolving hybrid-HTML attempts to expand its functionality by adding simple scripting capabilities to HTML's original descriptive tags and of creating an "open" environment for the integration of third-party components.

BSPR:

However, the hybrid-HTML format has a number of limitations and disadvantages. For example, since hybrid-HTML mixes procedural scripts with descriptive markup tags, a scripting code is required in order to control the behavior of objects within the HTML document. Thus, HTML publishers must understand and master procedural programming in order to take advantage of HTML's newest features. Moreover, critical semantic information is now embedded in procedural code. These procedural "scripts" cannot be inspected or parsed by generic, SGML tools and often contain

BSPR:

instructions proprietary to a particular browser vendor.

BSPR:

Another disadvantage is that, in the hybrid-HTML format, each component "plug-in" is a separate and independent "black box" with its own, idiosyncratic interface to

the browser's scripting language. Even components developed in a common language, such as Java-based applets, are controlled through the particular interface exposed by a specific applet. Thus, it is difficult, particularly for non-programmers, to integrate a variety of components into a single component-based application.

BSPR:

Yet another problem is that, although HTML tags can be used to declare attributes associated with standard browser components and user-interface controls, there is little or no coordination between the structure of the markup tags and the underlying data structures employed by the browser application itself. For example, HTML elements do not "inherit" properties from "superclass" elements. Thus, hybrid-HTML does not provide a sufficiently sophisticated architecture to integrate large-scale component-based applications.

BSPR:

Finally, hybrid-HTML is limited by the rigidity of HTML itself. By design, HTML only allows for a fixed set of element and attribute tags. There is no provision for developers to add new functionality through application-specific element tags or attributes. Nor is there any ability to map new element tags to new components or to extend the functionality of the browser application through code-based subclassing. Fundamentally, HTML was designed and is still used as a document markup language that is "processed" by an autonomous application: the browser. The intent and design of HTML was to provide a standardized format for marking up "multimedia" documents. All sophisticated application processing is delegated to external resources, i.e. scripting languages, embedded "applets", server-side applications, and the like. Consequently, HTML-based browsers were not designed to actually create and configure a component-based application through descriptive tags alone.

BSPV:

(5) script-oriented "authoring tools" such as Apple's.RTM. Hypertalk or Macromedia's.RTM. Director.RTM., which provide user-friendly environments for developing multimedia applications using the tools' built-in functionality and single-purpose, proprietary scripting languages.

DEPR:

Thereafter, the element processor 118 calls 452 any initialization method for the current component 212. In one embodiment, the initialization method may be specified in an section of the corresponding element 306. Alternatively, a default method, such as init(), may be used, as in the standard Java Applet class.

DEPR:

In one embodiment of the invention, after all of the components 212 have been processed, the init() and start() methods of the root component 212 are invoked in order to begin actual execution of the application 214. Preferably, the root component 212 implements the interface, MinAppIntf which is the minimal interface that is expected for any root-level component 212 to implement in order to be correctly instantiated and launched. In one embodiment, the MinAppIntf's interface is patterned after Java's standard Applet class. An example of the interface is provided below in Java pseudo-code:

DETL:

They may have a Type attribute 310. They may have an ID attribute 310 (called "Name" in BML) used to refer to that bean in expressions elsewhere. Any time it needs to refer to itself, it may use the ".sub.-- SELF" ID. They may have a Value attribute 310 which specifies a literal value for the bean 212. When this is present, the Class attribute 310 of its TYPE is used to convert the Value to the actual bean 212 for that tag 314 using ADML's built-in conversion mechanism 126. This attribute is not permitted when there is a VALUE child tag 314. They may have a VALUE child tag 314 which contains an expression for creating the bean 212, if the default constructor for the bean 212 is not sufficient. This is described in more detail below. No forward references are allowed here (i.e. objects below the closing tag 314 of the parent of the VALUE tag are not visible yet. Because children of the parent are processed before the parent, ID's of the VALUE's preceding sibling tags are visible). This tag 314 is not permitted when there is a Value attribute 310 for the bean 212. They may have an INIT section, which is used to perform initialization such as setting bean properties 320. Unlike BML, forward references are possible here (i.e. all tag

ID's are visible). They may have START and STOP sections. These are used when the parent frame gets a start and stop message. Typically, if the ADML page were inside an Applet, they would be connected to its start() and stop() methods. Inside an application, START is invoked whenever the frame becomes visible, and STOP is invoked whenever the frame becomes invisible. They may have EVENT sections. An EVENT section has a Type attribute 310 which specifies the desired listener method to which the ADML runtime should respond. ADML has built-in listeners for all AWT event types as well as all JFC events. New ones can be registered with the LISTENER tag in the HEAD section. The event types are the names of the listener methods a listener. For example, "mousePressed" and "mouseReleased" are two event types in ADML, as are all of the other method names from the java.awt.event.MouseListener interface.

WEST **Generate Collection**

L21: Entry 24 of 29

File: USPT

Mar 7, 2000

DOCUMENT-IDENTIFIER: US 6035119 A

TITLE: Method and apparatus for automatic generation of text and computer-executable code

ABPL:

The present invention provides a method, apparatus, and medium for adding text and text-based components to a Web page hosted on a server. A control, which is run at the designing time of the web page (design-time), when implemented, writes HTML information to a created web page. The created HTML information may include text and other text based components (client and server scripting, applets, ActiveX controls, Java scripting, and other components). Through the use of OLE, the controls incorporate author-friendly capabilities including in-place editing, property sheets, and persistence. Through the use of these controls, authors may automate the web page generation process and eliminate redundant coding.

BSPR:

In general, the present invention relates to authoring of text and computer-executable code, and more particularly to techniques for automatically generating HTML text and script within web pages.

BSPR:

One such example of the improvements in Web technology are dynamic web pages which can provide updated information to the user when the web page is downloaded. For example, dynamic web pages may be used to provide time of day information, table of contents information and searching capabilities of the web site. One common way of providing these dynamic web pages is the use of WebBot components found in the FrontPage web page authoring and management software (from the Microsoft Corporation of Redmond, Wash.). WebBot components may comprise run-time scripts or pointers to additional pages. WebBots, which can be dynamic components, when activated, process information present on the FrontPage web at the time that the WebBots components were activated and may generate HTML text on the web page. WebBot components are implemented using a dynamic link library (DLL).

BSPR:

WebBot components, however, are limited in that they cannot take advantage of property browsers and property pages. In addition, certain WebBot components require FrontPage Server Extensions on the web server in which the FrontPage web is published. While the advanced WebBot components perform dynamic processing when the web page is requested, only servers which have the FrontPage Server Extensions may perform this processing.

BSPR:

Java applets and certain web browser plug-ins may also be implemented to provide dynamic web pages. However, in a simpler form, Java applets and web browser plug-ins are limiting in that they must be written in only certain programming languages.

BSPR:

Dynamic web pages may also be provided using ActiveX controls developed by Microsoft Corporation of Redmond, Wash. ActiveX controls are a popular and effective way of encapsulating functionality and distributing this functionality across a variety of platforms. Unlike, Java applets and certain plug-ins, ActiveX controls may be written in any number of programming languages. ActiveX controls are based on OLE controls or OCX controls and are components (or objects) which can be inserted into a web page or other application to reuse packaged functionality which someone else programmed. For example, Microsoft's Internet Explorer 3.0 contains ActiveX controls which allow users to enhance web pages with sophisticated formatting features and animation. ActiveX controls are an enhancement over Java applets and certain web browser plug-ins since ActiveX

controls may also be used in applications written in any number of programming languages. Hundreds of ready to use ActiveX controls are now available with functionality ranging from a timer control (which simply notifies the system at a particular time) to full-featured spreadsheets and words processors. Web authors may also write their own ActiveX controls using Visual Basic 5.0 or Visual C++ and an ActiveX control framework such as Microsoft Foundation Class Library (MFC), the ActiveX Template Library (ATL), or the BaseClt framework. ActiveX controls are designated within a web page by use of a standard HTML tag. Typically, object tags include a set of parameters which include name value pairs, which are the persistence information of the control. The object tag includes a set of parameters that specify which control should be used and control the appearance and behavior of the control.

BSPR:

ActiveX controls currently, however, only are compatible with Internet Explorer (IE) web browsers (by the Microsoft Corporation). While developers desire to use the encapsulation and distribution advantages of ActiveX controls, they may wish their developed code to run in browsers other than Internet Explorer. Developers are faced with a dilemma: to author full, rich code using all the features of ActiveX controls while limiting the end use of the controls to only Internet Explorer or to author simple code which can run on a variety of browsers. ActiveX controls are also limiting in that they cannot author HTML and script in a web page. Web developers must author HTML and script themselves to provide certain features on a web page. Further, the HTML and script must be authored in alternative representations in order to accommodate all types of web browsers. This forces web page authors to become experts in the capabilities and needs of all web browsers.

BSPR:

Many of the aforementioned problems are solved by providing a method and apparatus for authoring text and computer-executable code. In one embodiment of the present invention, the present invention is a tool for web developers which automatically generates HTML text and script onto a web page. Under this embodiment, a developer when designing a web page inserts a control into the web page. When the file is saved, the editor or container asks the control for both its design-time information as well as its run-time HTML text and run-time script which should be written into the file. The design-time information of the control is then rendered invisible to the run-time user by being wrapped inside an HTML comment. The control is thereby made invisible to any downstream processor that respects HTML comments such as web browsers. In an alternative embodiment, the design-time information may be filtered out prior to processing the run-time information. This filter may also be referred to as stripping the design-time information of the control out of the web page when forwarded to the run-time platform. The present invention may be implemented within any system to generate text and/or script. For example, the present invention may be used to generate C++ code in a C++ development environment.

BSPR:

Advantageously, in one embodiment, the present invention allows developers to design web pages that can be accessed by web browsers regardless of their handling capabilities such as ActiveX or HTML only. In addition, processing is performed when the files are saved rather than when the file is accessed by the web browser.

BSPR:

In one application, the present invention generates HTML script and Active Server Page (ASP) code that is inserted into an ASP page. In this case, the control is stripped from the file when the file is processed for delivery to a web browser.

BSPR:

In one embodiment, the present invention uses a Component Object Model (COM) component thereby allowing the controls to be utilized across multiple applications. The controls may be, for example, ActiveX controls that automatically generate HTML text and script when the file is saved. The controls may be written using any development tool including, but not limited to, Visual C++ 5.0, Visual Basic 5.0, Borland Delphi or C++, and Symantec C++. Advantageously, the present invention may be implemented within any type of HTML editor, including but not limited to, FrontPage and Visual InterDev editors.

DEPR:

The present invention is described as it would be used for Internet applications, which is one embodiment of the invention. The present invention, however, may be implemented generally within any number of computer applications including generally any software development tool for the authoring of text and computer-executable code. In the context of developing web pages, the present invention provides web content developers a tool for authoring hypertext markup language (HTML) text and script. In this context, the present invention may be implemented within an authoring tool such as, for example, Visual C++, Visual InterDev and Visual J++ by Microsoft Corporation of Redmond, Wash.

DEPR:

Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules. Generally, program modules include routines, programs, objects, scripts, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with any number of computer system configurations including, but not limited to, distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices. The present invention may also be practiced in personal computers (PCs), hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like.

DEPR:

FIG. 1 is a schematic diagram depicting the general operation of the present invention in accordance with one embodiment of the invention. A file 115 designating a web page may be developed using an editor 110. The present invention may be implemented by any number of editors presently available and future versions. Available editors include, for example, Visual InterDev and FrontPage (by the Microsoft Corporation). The developer may insert a design-time control 120 into the file 115 to provide a desired functionality to the web page. The design-time control 120 may be any control capable of generating text and/or computer-executable code. Such code or text may be generated when the file 115 containing the design-time control 120 is saved. Alternatively, it may be generated at other times including, for example, when the control 120 is inserted in the file 115 (or document depending on the application used). In one embodiment, the design-time control 120 is an ActiveX control that has an additional Component Object Model (COM) interface that generates HTML text and script.

DEPR:

When the developer wishes to save the file 115, the file 115 is typically saved on a web server 130 as web page 135. When saved to web page 135, the file 115 in editor 110 is first processed to generate the appropriate run-time HTML text or script 140 in accordance with the design-time control 120. The design-time control 120 is then placed within an HTML comment 150 to render the design-time information of the control 120 invisible at run-time to the run-time process. The run-time text or script 140 may have any content including, but not limited to, server-side scripting intended to be consumed by Active Server Pages, client-side scripting to be consumed by a browser, HTML tags and content. When the web page 135 is retrieved by a web browser 160, only the run-time HTML text or script 140 is processed by the web browser 160. The design-time control 120 is ignored by the web browser 160 since it is wrapped inside the HTML comment 150. Alternatively, the web server 130 may filter or strip the HTML comment 150 from the web page 135 so that the design-time control 120 is prevented from being sent to the browser 160. The present invention thereby provides web site developers a convenient tool for authoring of dynamic web pages. The present invention provides developers controls which can be inserted into web pages to provide desired functionality. A developer may include previously designed design-time controls within a page thereby alleviating the burden on the developer from having to manually author all web page content. Repeatable web page content may be encapsulated into component objects which can be used to easily author desired text or related code. The design-time control allows this repeatable web page content to be componentized for later reuse. Through the execution of the design-time control during, for example, the saving process, the run-time text is created and the design-time control commented out. In addition, while previously developers were required to

manually replicate their web content in alternative representations so as to accommodate all types of web browsers, with the present invention, developers may author web content and allow the replication of the alternative formats to be handled by the design-time controls.

DEPR:

Under one embodiment, when a web site developer is authoring a web page, namely during design-time, the present invention operates similar to ActiveX controls. Controls, in general, are reusable software components that can be embedded in the context of their container and contribute additional capabilities and user interface to their container. Controls may, for example, when embedded within a web page, present a graphical user interface within the area of the control (where located on the web page) when displayed on a browser. Controls are capable of handling their own drawing, accepting mouse and keyboard inputs, providing content menus and other functions which generally extend the graphical editing environment of a container. In addition, controls can use a property browser and property page frames of their container to allow the user to set properties of the control. Additionally, controls are predefined and installed programs which may be accessed by a variety of different applications.

DEPR:

Through the use of this interface, the design-time control may generate text for inclusion into a page. As all HTML content is based on text, the design-time control which controls the insertion of text has significant advantages, at least, as a tool for targeting text-using clients, independent of the type of client's browser.

DEPR:

One may use design-time controls to author any text-based solution including HTML tags and content, language-independent server and client scripting, Java applets, ActiveX controls, ActiveX server components and other text-based solutions. One embodiment of the invention includes third parties hosting design-time controls which are downloaded as needed to an author's editing environment.

DEPR:

It will be noted that the run-time text is capable of handling any number of run-time implementations, including but not limited to HTML tags, server components, Java applets, Netscape plug-ins, or script. At step 430, the design-time control is ended by a metadata comment that includes both the TYPE="DesignerControl" and the endspan attribute: (). The following depicts an example of the run-time text in relation to the design-time control as they are persisted within an HTML comment:

DEPR:

In the case where the control is within an Active Server Page (ASP), the design-time control is prevented from being delivered to the client browser. When the Active Server (hosting the ASP) encounters a metadata comment, the server strips the comment from the file before processing, leaving behind only the run-time text generated by the design-time control. This occurs before the performance of any other logic, making the run-time text generated by the design-time control fully active.

DEPR:

Finally, when a client (including, for example, a web browser) retrieves the files at run-time, the design-time control information is ignored since it is wrapped inside an HTML comment and displays the run-time text. In the case where the file is processed by an active server, the control data is stripped from the file before it is processed. When the active server sees a metadata comment, it will strip the comment from the file before processing. This leaves behind just the run-time text of the design-time control. This step occurs before any other logic so the run-time text is fully active. Thus, for example, the following describes a file before it is processed by an active server:

DEPR:

Depending on its programming, each design-time may include non-ActiveX scripts as well. So, by instantiating the design-time control into a web page, the resulting run-time code may include both the ActiveX implementation as well as the non-ActiveX counterpart. Non-ActiveX script may include, for example, Java (by Sun Microsystems). Other non-ActiveX scripts may also be included.

CLPR:

4. The computer-implemented method of claim 1, wherein said third set of instructions include script for controlling activities of a computer.

CLPR:

9. The method of claim 8, wherein said text information is interpreted as script for implementation on at least one of said server and said client computer.

CLPR:

16. The computer-readable medium of claim 13, wherein said third set of instructions include script for controlling activities of a computer.

CLPR:

23. The computer system of claim 20, wherein said third set of instructions include script for controlling activities of a computer.

WEST **Generate Collection**

L21: Entry 26 of 29

File: USPT

Sep 28, 1999

DOCUMENT-IDENTIFIER: US 5958008 A

TITLE: Software system and associated methods for scanning and mapping dynamically-generated web documents

ABPL:

A visual Web site analysis program, implemented as a collection of software components, provides a variety of features for facilitating the analysis and management of Web sites and Web site content. A mapping component scans a Web site over a network connection and builds a site map which graphically depicts the URLs and links of the site. Site maps are generated using a unique layout and display methodology which allows the user to visualize the overall architecture of the Web site. Various map navigation and URL filtering features are provided to facilitate the task of identifying and repairing common Web site problems, such as links to missing URLs. A dynamic page scan feature enables the user to include dynamically-generated Web pages within the site map by capturing the output of a standard Web browser when a form is submitted by the user, and then automatically resubmitting this output during subsequent mappings of the site. The Web site analysis program is implemented using an extensible architecture which includes an API that allows plug-in applications to manipulate the display of the site map. Various plug-ins are provided which utilize the API to extend the functionality of the analysis program, including an action tracking plug-in which detects user activity and behavioral data (link activity levels, common site entry and exit points, etc.) from server log files and then superimposes such data onto the site map.

BSPR:

In accordance with another aspect of the invention, the Web site analysis program is based on an extensible architecture that allows software components to be added that make extensive use of the program's mapping functionality. Specifically, the architecture includes an API (application program interface) which includes API procedures ("methods") that allow other applications ("plug-ins") to, among other things, manipulate the display attributes of the nodes and links within a site map. Using these methods, a plug-in application can be added which dynamically superimposes data onto the site map by, for example, selectively modifying display colors of nodes and links, selectively hiding nodes and links, and/or attaching alphanumeric annotations to the nodes and links. The API also includes methods for allowing plug-in components to access Web site data (both during and following the Web site scanning process) retrieved by the scanning routines, and for adding menu commands to the user interface of the main program.

BSPR:

In accordance with another aspect of the invention, software routines (preferably implemented within a plug-in application) are provided for processing a Web site's server access log file to generate Web site usage data, and for displaying the usage data on a site map. This usage data may, for example, be in the form of the number of "hits" per link, the number of Web site exit events per node, or the navigation paths taken by specific users ("visitors"). This usage data is preferably generated by processing the entries within the log file on a per-visitor basis to determine the probable navigation path taken by each respective visitor to the Web site. (Standard-format access log files which record each access to any page of the Web site are typically maintained by conventional Web servers.) In a preferred implementation, the usage data is then superimposed onto the site map (using the API methods) using different node and link display colors to represent different respective levels of user activity. Using this feature, Webmasters can readily detect common "problem areas" such as congested links and popular Web site exit points. In addition, by looking at individual navigation paths on a per-visitor basis, Webmasters can identify popular navigation paths taken by visitors to the site.

BSPR:

To effectuate the capture of one or more datasets in the preferred implementation, the user initiates a capture session from the user interface; this causes a standard Web browser to be launched and temporarily configured to use the Web site analysis program as an HTTP-level proxy to communicate with Web sites. Thereafter, until the capture session is terminated by the user, any pages retrieved with the browser, and any forms (datasets) submitted from the browser, are automatically recorded by the Web site analysis program into the site map. When the site map is subsequently updated (using an "automatic update" option of the user interface), the scanning routines automatically re-enter the captured datasets into the corresponding forms and recreate the form submissions. The dynamically-generated Web pages returned in response to these automatic form submissions are then added to the updated site map as respective nodes. A related aspect of the invention involves the associated method of locally capturing the output of the Web browser to generate a sequence that can subsequently be used to automatically evaluate a Web site.

DEPR:

Document. Generally, a collection of data that can be viewed using an application program, and that appears or is treated as a self-contained entity. Documents typically include control codes that specify how the document content is displayed by the application program. An "HTML document" is a special type of document which includes HTML (HyperText Markup Language) codes to permit the document to be viewed using a Web browser program. An HTML document that is accessible on a World Wide Web site is commonly referred to as a "Web document" or "Web page." Web documents commonly include embedded components, such as GIF (Graphics Interchange Format) files, which are represented within the HTML coding as links to other URLs. (See "HTML" and "URL" below.)

DEPR:

World Wide Web. A distributed, global hypertext system, based on an set of standard protocols and conventions (such as HTTP and HTML., discussed below), which uses the Internet as a transport mechanism. A software program which allows users to request and view World Wide Web ("Web") documents is commonly referred to as a "Web browser," and a program which responds to such requests by returning ("serving") Web documents is commonly referred to as a "Web server."

DEPR:

Content Object. As used herein, a data entity (document, document component, etc.) that can be selectively retrieved from a web site. In the context of the World Wide Web, common types of content objects include HTML documents, GIF files, sound files, video files, Java applets and aglets, and downloadable applications, and each object has a unique identifier (referred to as the "URL") which specifies the location of the object. (See "URL" below.)

DEPR:

HTML (HyperText Markup Language). A standard coding convention and set of codes for attaching presentation and linking attributes to informational content within documents. During a document authoring stage, the HTML codes (referred to as "tags") are embedded within the informational content of the document. When the Web document (or "HTML document") is subsequently transmitted by a Web server to a Web browser, the codes are interpreted by the browser and used to parse and display the document. In addition to specifying how the Web browser is to display the document, HTML tags can be used create hyperlinks to other Web documents. For more information on HTML, see Ian S. Graham, The HTML Source Book, John Wiley and Sons, Inc., 1995 (ISBN 0471-11894-4).

DEPR:

HTTP (Hypertext Transfer Protocol). The standard World Wide Web client-server protocol used for the exchange of information (such as HTML documents, and client requests for such documents) between a Web browser and a Web server. HTTP includes several different types of messages which can be sent from the client to the server to request different types of server actions. For example, a "GET" message, which has the format GET , causes the server to return the content object located at the specified URL.

DEPR:

CGI (Common Gateway Interface). A standard interface which specifies how a Web server (or possibly another information server) launches and interacts with

external programs (such as a database search engine) in response to requests from clients. With CGI, the Web server can serve information which is stored in a format that is not readable by the client, and present such information in the form of a client-readable Web page. A CGI program (called a "CGI script") may be invoked, for example, when a Web user fills out an on-screen form which specifies a database query. For more information on CGI, see Ian S. Graham, *The HTML Source Book*, John Wiley and Sons, Inc., 1995 (ISBN 0471-11894-4), pp. 231-278.

DEPR:

OLE (Object Linking and Embedding). An object technology, implemented by Windows-based applications, which allows objects to be linked to one another and embedded within one another. OLE Automation, which is a feature of OLE 2, enables a program's functionality to be exposed as OLE objects that can be used to build other applications. For additional information on OLE and OLE Automation, see OLE 2 Programmer's Reference Manual, Volume One, Microsoft Corporation, 1996 (ISBN 1-55615-628-6).

DEPR:

Given the address of a Web site's home page, Astra automatically scans the Web site and creates a graphical site map showing all of the URLs of the site and the links between these URLs. In accordance with one aspect of the invention, the layout and display method used by Astra for generating the site map provides a highly intuitive, graphical representation which allows the user to visualize the layout of the site. Using this mapping feature, in combination with Astra's powerful set of integrated tools for navigating, filtering and manipulating the Web site map, users can intuitively perform such actions as isolate and repair broken links, focus in on Web pages (and other content objects) of a particular content type and/or status, and highlight modifications made to a Web site since a prior mapping. In addition, users can utilize a Dynamic Scan.TM. feature of Astra to automatically append dynamically-generated Web pages (such as pages generated using CGI scripts) to their maps. Further, using Astra's activity monitoring features, users can monitor visitor activity levels on individual links and URLs, and study visitor behavior patterns during Web site visits.

DEPR:

In accordance with another aspect of the invention, Astra has a highly extensible architecture which facilitates the addition of new tools to the Astra framework. As part of this architecture, a "core" Astra component (which includes the basic Web site scanning and mapping functionality) has an API for supporting the addition of plug-in components. This API includes functions for allowing the plug-in components to manipulate the display of the site map, and to display their own respective data in conjunction with the Astra site map. Through this API, new applications can be added which extend the functionality of the package while taking advantage of the Astra mapping scheme.

DEPR:

FIG. 1 illustrates a site map 30 of a demonstration Web site which was derived from the actual Web site of Mercury Interactive, Inc. (i.e., the URLs accessible under the "merc-int.com" Internet domain name). (For purposes of this detailed description, it may be assumed that "Web site" refers to the content associated with a particular Internet domain name.) The Web site is depicted by Astra as a collection of nodes, with pairs of nodes interconnected by lines. Each node of the map represents a respective content object of the Web site and corresponds to a respective URL. (The term "URL" is used herein to refer interchangeably to both the address of the content object and to the object itself; where a distinction between the two is helpful to an understanding of the invention, the term "URL" is followed by an explanatory parenthetical.) Examples of URLs (content objects) which may exist within a typical Web site include HTML documents (also referred to herein as "Web pages"), image files (e.g., GIF and PCX files), mail messages, Java applets and aglets, audio files, video files, and applications.

DEPR:

The lines which interconnect the nodes (URL icons) in FIGS. 1-3 (and the subsequent figures with screen displays) represent links between URLs. As is well understood in the art, the functions performed by these links vary according to URL type. For example, a link from one HTML document to another HTML document normally represents a hyperlink which allows the user to jump from one document to the other while navigating the Web site with a browser. In FIG. 1, an example of a hyperlink which links the home page URL (shown at the center of the map) to

another HTML page (displayed to the right of the home page) is denoted by reference number 32. (As generally illustrated in FIG. 1 and the other figures which illustrate screen displays, regular HTML documents are displayed by Astra as a shaded document having text thereon.) A link between an HTML document and a GIF file, such as link 36 in FIG. 3, normally represents a graphic which is embedded within the Web page.

DEPR:

Maps of the type illustrated in FIG. 1 are generated by Astra using an HTTP-level scanning process (described below) which involves the reading and parsing the Web site's HTML pages to identify the architecture (i.e., the arrangement of URLs and links) of the Web site, and to obtain various status information (described below) about the Web site's URLs. The basic scanning process used for this purpose is generally similar to the scanning process used by conventional Webcrawlers. As part of Astra's Dynamic Scan feature, Astra additionally implements a special dynamic page scanning process which permits dynamically-generated Web pages to be scanned and included in the Web site map. As described below, this process involves capturing the output of a Web browser when the user submits an HTML-embedded form (such as when the user submits a database query), and then reusing the captured dataset during the scanning process to recreate the form submission and append the results to the map.

DEPR:

While navigating the map, the user can retrieve a URL (content object) from the server by double-clicking on the corresponding URL icon; this causes Astra to launch the client computer's default Web browser (if not already running), which in-turn retrieves the URL from the Web server. For example, the user can double-click on the URL icon for an HTML document (using the left mouse button) to retrieve and view the corresponding Web page. When the user clicks on a URL icon using the right mouse button, a menu appears which allows the user to perform a variety of actions with respect to the URL, including viewing the URL's properties, and launching an HTML editor to retrieve and edit the URL. With reference to FIG. 3, for example, the user can click on node 44 (using the right mouse button), and can then launch an HTML editor to edit the HTML document and delete the reference to missing URL 45. (As illustrated by FIG. 3, missing URLs are represented within Astra maps by a question mark icon.)

DEPR:

Another benefit of this site map layout and display methodology is that the resulting display structure is well suited for the overlaying of information on the map. Astra takes full advantage of this benefit by providing a set of API functions which allow other applications (Astra plug-ins) to manipulate and add their respective display data to the site map. An example of an Astra plug-in which utilizes this feature is the Action Tracker.TM. tool, which superimposes user activity data onto the site map based on analyses of server access log files. The Astra plug-in API and the Action Tracker plug-in are described in detail below.

DEPR:

As illustrated in FIG. 1, the Astra menu bar includes seven menu headings: FILE, VIEW, SCAN, MAP, URL, TOOLS and HELP. From the FILE menu the user can perform various file-related operations, such as save a map file to disk or open a previously generated map file. From the VIEW menu the user can select various display options of the Astra GUI. From the SCAN menu the user can control various scanning-related activities, such as initiate or pause the automatic updating of a map, or initiate a dynamic page scan session. From the MAP menu, the user can manipulate the display of the map, by, for example, collapsing (hiding) all leaf nodes, or selecting the Visual Web Display mode. From the URL menu, the user can perform operations with respect to user-selected URLs, such as display the URL's content with a browser, invoke an editor to modify the URL's content, and display the incoming or outgoing links to/from the URL.

DEPR:

FIG. 7 pictorially illustrates the general architecture of Astra, as installed on a client computer 92. As illustrated, the architecture generally consists of a core Astra component 94 which communicates with a variety of different Astra plug-in applications 96 via a plug-in API 98. The Astra core 94 includes the basic functionality for the scanning and mapping of Web sites, and includes the above-described GUI features for facilitating navigation of Web site maps. Through

the plug-in API 98, the Astra core 94 provides an extensible framework for allowing new applications to be written which extend the basic functionality of the Astra core. As described below, the architecture is structured such that the plug-in applications can make extensive use of Astra site maps to display plug-in specific information.

DEPR:

The Astra plug-ins 96 and API 98 are based on OLE Automation technology, which provides facilities for allowing the plug-in components to publish information to other objects via the operating system registry (not shown). (The "registry" is a database used under the Windows.RTM. 95 and Windows.RTM. NT operating systems to store configuration information about a computer, including information about Windows-based applications installed on the computer.) At start-up, the Astra core 94 reads the registry to identify the Astra plug-ins that are currently installed on the client computer 92, and then uses this information to launch the installed plug-ins.

DEPR:

In a preferred implementation, the architecture includes five Astra plug-ins: Link Doctor, Action Tracker, Test World, Load Wizard and Search Meter. The functions performed by these plug-ins are summarized by Table 2. Other applications which will normally be installed on the client computer in conjunction with Astra include a standard Web browser (FIGS. 11 and 12), and one or more editors (not shown) for editing URL content.

DEPR:

The Astra API allows external client applications, such as the plug-in applications 96 shown in FIG. 7, to communicate with the Astra core 94 in order to form a variety of tasks. Via this API, client applications can perform the following types of operations:

DEPR:

As is conventional with Internet applications, the Astra core 94 uses the TCP/IP layer 108 of the computer's operating system to communicate with the Web site 113. Any one or more of the Astra plug-ins 96 may also use the TCP/IP layer 108 to communicate with the Web site 113. In the preferred embodiment, for example, the Action Tracker plug-in communicates with the Web sites (via the Astra plug-in API) to retrieve server access log files for performing Web site activity analyses.

DEPR:

Each Node object 115 represents a respective node (URL) of the site map, and each Edge object 116 represents a respective link between two URLs (nodes) of the map. Associated with each Node object and each Edge object is a set of attributes (not shown), including display attributes which specify how the respective object is to be represented graphically within the site map. For example, each Node object and each Edge object include respective attributes for specifying the color, visibility, size, screen position, and an annotation for the display of the object. These attributes can be manipulated via API calls to the methods supported by these objects 115, 116. For example, the Astra plug-ins (FIG. 7) can manipulate the visibility attributes of the Edge objects to selectively hide the corresponding links on the screen. (This feature is illustrated below in the description of the Action Tracker plug-in.) In addition, the Astra API includes methods for allowing the plug-ins to define and attach custom attributes to the Edge and Node objects.

DEPR:

The methods of the Astra plug-in API generally fall into five functional categories. These categories, and the objects to which the associated methods apply, are listed below. Additional information on these methods is provided in the API listing in Appendix B.

DEPR:

As will be appreciated from the foregoing, the Astra architecture provides a highly extensible mapping framework which can be extended in functionality by the addition of new plug-ins applications. Additional aspects of the architecture are specified in the API description of Appendix B.

DEPR:

A feature of the invention which permits the scanning and mapping of

dynamically-generated Web pages will now be described. By way of background, a dynamically-generated Web page ("dynamic page") is a page that is generated "on-the-fly" by a Web site in response to some user input, such as a database query. Under existing Web technology, the user manually types-in the information (referred to herein as the "dataset") into an embedded form of an HTML document while viewing the document with a Web browser, and then selects a "submit" type button to submit the dataset to a Web site that has back-end database access or real-time data generation capabilities. (Technologies which provide such Web server extension capabilities include CGI, Microsoft's ISAPI, and Netscape's NSAPI.) A Web server extension module (such as a CGI script) then processes the dataset (by, for example, performing a database search, or generating real-time data) to generate the data to be returned to the user, and the data is returned to the browser in the form of a standard Web page.

DEPR:

One deficiency in existing Web site mapping programs is that they do not support the automatic retrieval of dynamic pages. As a result, these mapping programs are not well suited for tracking changes to back-end databases, and do not provide an efficient mechanism for testing the functionality of back-end database search components. The present invention overcomes these deficiencies by providing a mechanism for capturing datasets entered by the user into a standard Web browser, and for automatically re-submitting such datasets during the updating of site maps. The feature of Astra which provides these capabilities is referred to as Dynamic Scan.TM..

DEPR:

FIG. 11 illustrates the general flow of information between components during a Dynamic Scan capture session, which can be initiated by the user from the Astra tool bar. Depicted in the drawing is a client computer 92 communicating with a Web site 113 over the Internet 110 via respective TCP/IP layers 108, 178. The Web site 113 includes a Web server application 112 which interoperates with CGI scripts (shown as layer 180) to generate Web pages on-the-fly. Running on the client computer 92 in conjunction with the Astra application 94 is a standard Web browser 170 (such as Netscape Navigator or Microsoft's Internet Explorer), which is automatically launched by Astra when the user activates the capture session. As illustrated, the Web browser 170 is configured to use the Astra application 94 as an HTTP-level proxy. Thus, all HTTP-level messages (client requests) generated by the Web browser 170 are initially passed to Astra 94, which in-turn makes the client requests on behalf of the Web browser. Server responses (HTML pages, etc.) to such requests are returned to Astra by the client computer's TCP/IP layer 108, and are then forwarded to the browser to maintain the impression of normal browsing.

DEPR:

During the Dynamic Scan capture session, the user types-in data into one or more fields 174 of an HTML document 172 while viewing the document with the browser 170. The HTML document 172 may, for example, be an internal URL which is part of a Web site map, or may be an external URL which has been linked to the site map for mapping purposes. When the user submits the form, Astra extracts the manually-entered dataset, and stores this dataset (in association with the HTML document 172) for subsequent use. When Astra subsequently re-scans the HTML document 172 (during an Automatic Update of the associated site map), Astra automatically retrieves the dataset, and submits the dataset to the Web site 113 to recreate the form submission. Thus, for example, once the user has typed-in and submitted a database query in connection with a URL of a site map, Astra will automatically perform the database query (and map the results, as described below) the next time an Automatic Update of the map is performed.

DEPR:

With further reference to FIG. 11, when the Web site 113 returns the dynamic page during the capture session (or during a subsequent Automatic Update session), Astra automatically adds a corresponding node to the site map, with this node being displayed as being linked to the form page. (Screen displays taken during a sample capture session are shown in FIGS. 13-15 and are described below.) In addition, Astra parses the dynamic page, and adds respective nodes to the map for each outgoing link of the dynamic page. (In the default setting, these outgoing links are not scanned.) Astra also parses any static Web pages that are retrieved with the browser during the Dynamic Scan capture session, and updates the site map (by appending appropriate URL icons) to reflect the static pages.

DEPR:

Prior to initiating the Dynamic Scan session, the user specifies a page 172 which includes an embedded form. (This step is not shown in FIG. 12). This can be done by browsing the site map with the Astra GUI to locate the node of a form page 172 (depicted by Astra using a special icon), and then selecting the node with the mouse. The user then initiates a Dynamic Scan session, which causes the following dialog to appear on the screen: YOU ARE ABOUT TO ENTER DYNAMIC SCAN MODE. IN THIS MODE YOU WORK WITH A BROWSER AS USUAL, BUT ALL YOUR ACTIONS (INCLUDING FORM SUBMISSIONS) ARE RECORDED IN THE SITE MAP. TO EXIT FROM THIS MODE, PRESS THE "STOP DYNAMIC SCAN" BUTTON ON THE MAIN TOOLBAR OR CHOOSE THE "STOP DYNAMIC SCAN" OPTION IN THE SCAN MENU.

DEPR:

When the user clicks on the "OK" button, Astra modifies the configuration of the Web browser 170 within the registry 182 of the client computer to set Astra 94 as a proxy of the browser, as illustrated by arrow A of FIG. 12. (As will be recognized by those skilled in the art, the specific modification which needs to be made to the registry 182 depends upon the default browser installed on the client computer.) Astra then launches the browser 170, and passes the URL (address) of the selected form page to the browser for display. Once the browser has been launched, Astra modifies the registry 182 (arrow B) to restore the original browser configuration. This ensures that the browser will not attempt to use Astra as a proxy on subsequent browser launches, but does disable the browser's use of Astra as a proxy during the Dynamic Scan session.

DEPR:

As depicted in FIG. 12, the browser 170 retrieves and displays the form page 172, enabling the user to complete the form. In response to the submission by the user of the form, the browser 170 passes an HTTP-level (GET or POST) message to Astra 94, as indicated by arrow C. This message includes the dataset entered by the user, and specifies the URL (address) of the CGI script or other Web server extension component 180 to which the form is addressed. Upon receiving this HTTP message, Astra displays the dialog "YOU ARE ABOUT TO ADD A DATA SET TO THE CURRENT URL IN THE SITE MAP," and presents the user with an "OK" button and a "CANCEL" button.

DEPR:

Assuming the user selects the OK button, Astra extracts the dataset entered by the user and then forwards the HTTP-level message to its destination, as illustrated by arrow E. In addition, as depicted by arrow D, Astra stores this dataset in the Site Graph 114 in association with the form page 172. As described above, this dataset will automatically be retrieved and re-submitted each time the form page 172 is re-scanned as part of an Automatic Update operation. With reference to arrows F and G, when the Web server 112 returns the dynamic page 184, Astra 94 parses the page and updates the Site Graph 114 to reflect the page and any outgoing links of the dynamic page. (In this regard, Astra handles the dynamic page in the same manner as for other HTML documents retrieved during the normal scanning process.) In addition, as depicted by arrow H, Astra forwards the dynamic page 184 to the Web browser 170 (which in-turn displays the page) to maintain an impression of normal Web browsing.

DEPR:

Following the above sequence, the user can select the "stop dynamic scan" button or menu option to end the capture session and close the browser 170. Alternatively, the user can continue the browsing session and make additional updates to the site map. For example, the user can select the "back" button 186 (FIG. 14) of the browser to go back to the form page and submit a new dataset, in which case Astra will record the dataset and resulting page in the same manner as described above.

DEPR:

Although the system of the preferred embodiment utilizes conventional proxy technology to redirect and monitor the output of the Web browser 170, it will be recognized that other technologies and redirection methods can be used for this purpose. For example, the output of the Web browser could be monitored using conventional Internet firewall technologies.

DEPR:

FIG. 14 illustrates a subsequent screen display generated by starting a Dynamic Scan session with the Infoseek.TM. page selected, and then typing in the word "school" into the query field 194 of the page. (Intermediate displays generated by Astra during the Dynamic Scan session are omitted.) As illustrated in the figure, the Web browser comes up within a window 196, allowing the user to access the Astra controls and view the site map 190 during the Dynamic Scan session.

DEPR:

As generally illustrated by FIG. 15, in which the children 204 of the dynamic page 200 are represented with Astra's "not scanned," Astra does not automatically scan the children of the dynamically-generated Web page during the Dynamic Scan session. To effectively scan a child page 204, the user can retrieve the page with the browser during the Dynamic Scan session, which will cause Astra to parse the child page and update the map accordingly.

DEPR:

While the above-described Dynamic Scan feature is particularly useful in Web site mapping applications, it will be recognized that the feature can also be used to in other types of applications. For example, the feature can be used to permit the scanning of dynamically-generated pages by general purpose Webcrawlers. In addition, although the feature is implemented in the preferred embodiment such that the user can use a standard, stand-alone Web browser, it will be readily apparent that the feature can be implemented using a special "built-in" Web browser that is integrated with the scanning and mapping code.

DEPR:

An important feature of Astra is its the ability to track user (visitor) activity and behavior patterns with respect to a Web site and to graphically display this information (using color coding, annotations, etc.) on the site map. In the preferred embodiment, this feature is implemented in-part by the Action Tracker plug-in, which gathers user activity data by retrieving and analyzing server log files commonly maintained by Web servers. Using this feature, Webmasters can view site maps which graphically display such information as: the most frequently-accessed URLs, the most heavily traveled links and paths, and the most popular site entry and exit points. As will be appreciated by those skilled in the art, the ability to view such information in the context of a site map greatly simplifies the task of evaluating and maintaining Web site effectiveness.

DEPR:

FIG. 19 illustrates the general display format used by the Action Tracker plug-in to display activity levels on the links of a site. As illustrated by the screen display, the links between URLs are displayed using a color-coding scheme which allows the user to associate different link colors (and URL icon colors) with different relative levels of user activity. As generally illustrated by the color legend, three distinct colors are used to represent three (respective) adjacent ranges of user activity.

DEPR:

FIG. 20 illustrates the general process used by the Action Tracker plug-in to detect the link activity data (number of hits per link) from the log file. The displayed flow chart assumes that the log file has already been retrieved, and that the attribute "hits" has been defined for each link (Edge object) of the Site Graph and set to zero. As illustrated by the flow chart, the general decision process is applied line-by-line to the log file (each line representing an access to a URL) until all of the lines have been processed. With reference to blocks 250 and 252, each time a new line of the log file is ready, it is initially determined whether or not the log file reflects a previous access by the user to the Web site. This determination is made by searching for other entries within the log file which have the same user identifier (e.g., IP address) and an earlier date/time stamp.

DEPR:

Blocks 254 and 256 illustrate the steps that are performed if the user (visitor) previously visited the site. Initially, the Site Graph is accessed to determine whether a link exists from the most-recently accessed URL to the current URL, as indicated by decision block 254. If such a link exists, it is assumed that the visitor used this link to get to the current URL, and the usage level ("hits" attribute) of the identified link is incremented by one. If no such link is identified between the most-recently accessed URL and the current URL, an

assumption is made that the user back-tracked along the navigation path (by using the "BACK" button of the browser) before jumping to the current URL. Thus, decision step 254 is repeated for each prior access by the user to the site, in reverse chronological order, until either a link to the current URL is identified or all of the prior accesses are evaluated. If a link is detected during this process, the "hits" attribute of the link is incremented.

DEPR:

Usage Zones. When viewing a large site map in its entirety (as in FIG. 1), it tends to be difficult to identify individual URL icons within the map. This in-turn makes it difficult to view the color-coding scheme used by the Action Tracker plug-in to display URL usage levels. The Usage Zones.TM. feature alleviates this problem by enlarging the size of the colored URL icons (i.e., the icons of nodes which fall within the predetermined activity level thresholds) to a predetermined minimum size. (This is accomplished by increasing the "display size" attributes of these icons.) If these colored nodes are close together on the map, the enlarged icons merge to form a colored zone on the map. This facilitates the visual identification of high-activity zones of the site.

DEPR:

FIG. 22 illustrates the operation of Astra's Link Doctor plug-in. To access this feature, the user selects the "Link Doctor" option from the TOOLS menu while viewing a site map. The Link Doctor dialog box 284 then appears with a listing (in the "broken links" pane 286) of all of the broken links (i.e., URLs of missing content objects) detected within the site map. (Astra detects the missing links by searching the Site Graph for Node objects having a status of "not found.") When the user selects a URL from the broken links pane (as illustrated in the screen display), Astra automatically lists all of the URLs which reference the missing content object in the "appearing in" pane 288. This allows the user to rapidly identify all of the URLs (content objects) that are directly affected by the broken link.

DEPR:

While the present invention has been described in the context of a preferred software package for analyzing and managing web sites, many of the features of the invention can be applied to other types of web site analysis applications. For example, the dynamic page scanning features of the invention can be used to extend the functionality of Internet search engines, and the associated methods for capturing the output of a browser can be used to evaluate other aspects of Web sites. Accordingly, the breadth and scope of the present invention should be defined only in accordance with the following claims and their equivalents.

DEPL:

XI. Link Repair Plug-in (FIG. 22)

DEPU:

XI. Link Repair Plug-in

DETL:

TABLE 2 _____ PLUG-IN FUNCTION PERFORMED
Doctor Action Retrieves and evaluates server log files to generate Web site
Tracker activity data (such as activity levels on individual links), and
superimposes such data on site map in a user-adjustable manner. Test Generates and
drives tests automatically World Load Utilizes site map to automatically generate
test scripts for the Wizard load testing of Web sites with Mercury Interactive's
LoadRunner .TM. and SiteTest .TM. software packages. Search Displays search engine
results visually Meter _____

CLPR:

5. The method according to claim 1, wherein step (a) comprises launching a stand-alone browser program that is not a component of the web site analysis program.

CLPR:

6. The method according to claim 5, wherein step (a) further comprises dynamically configuring the stand-alone browser program to temporarily use the web site analysis program as a proxy.

CLPR:

15. The web site analysis program according to claim 14, wherein the dataset capture routine is configured to run in conjunction with a web browser program, and is configured to capture the dataset by extracting the dataset from a message generated by the web browser program, the message generated in response to a submission of the form by the user from the browser program.

CLPR:

16. The web site analysis program according to claim 15, wherein the dataset capture routine is configured to automatically launch the browser program in response to activation by the user of a dataset capture session from the user interface.

CLPR:

17. The web site analysis program according to claim 16, wherein the dataset capture routine is configured to operate as a proxy of the web browser program.

CLPR:

18. The web site analysis program according to claim 17, wherein the dataset capture includes executable code for temporarily configuring the web browser program to use the data capture routine as a proxy for communicating with web sites.

CLPR:

25. The method according to claim 24, further comprising the step of launching the browser program on the computer with the capture routine.

CLPR:

26. The method according to claim 25, wherein the step of launching the browser program is performed in response to activation by the user of a capture session from a user interface of the client program.

CLPR:

35. The method according to claim 34, wherein the step of capturing comprises temporarily configuring the browser program to use the analysis routine as a proxy for communicating with the web site.

CLPV:

receiving a dynamic page request message with the web site analysis program, the request message generated by the browser program in response to a submission by the user of the dataset; and

CLPV:

displaying the current version of the dynamically-generated document to the user with the browser program to create a user impression of normal browsing during the capture session.

CLPV:

(a) providing a capture routine which runs in conjunction with a browser program, the capture routine configured to capture datasets entered by users into forms of web documents;

CLPV:

(c) displaying a web document on a display screen of a computer with the browser program to allow a user to enter a dataset into a form of the web document;

CLPV:

(a) providing a data capture routine which runs in conjunction with a browser program on a computer, the computer connected to a network that interconnects the computer to the web site;

CLPV:

(c) capturing an output sequence of the browser with the data capture routine while a user interactively accesses the web site with the browser, and storing the output sequence within a memory of the computer, the output sequence including at least one web site address accessed by the user; and

CLPW:

(a) launching a browser program on the computer;

CLPW:

(b) displaying the web document on a display screen of the computer with the browser program to allow the user to enter the dataset into the form;